Chapter 2

# Relational Databases in Azure

## MICROSOFT EXAM OBJECTIVES COVERED IN THIS CHAPTER:

✓ Describe relational data workloads.

- Identify the right data offering for a relational workload.
- Describe relational data structures (e.g., tables, indexes, views).

✓ Relational Database Offerings in Azure.

- Describe Azure SQL database services such as Azure SQL Database, Azure SQL Managed Instance, and SQL Server on Azure Virtual Machine.
- Describe Azure Synapse Analytics.
- Describe Azure Database for PostgreSQL, Azure Database for MariaDB, and Azure Database for MySQL.

✓ Identify basic management tasks for relational data.

- Describe provisioning and deployment of relational data services.
- Describe method for deployment including the Azure portal, Azure Resource Manager templates, Azure PowerShell, and the Azure command-line interface (CLI).
- Identify data security components (e.g., firewall, authentication).
- Identify basic connectivity issues (e.g., accessing from on-premises, access with Azure VNets, access from Internet, authentication, firewalls).
- Identify query tools (e.g., Azure Data Studio, SQL Server Management Studio, sqlcmd utility, etc.).

✓ **Describe query techniques for data using SQL language.**

- **Compare Data Definition Language (DDL) versus Data Manipulation Language (DML).**
- **Query relational data in Azure SQL Database, Azure Database for PostgreSQL, and Azure Database for MySQL.**

Relational databases have been critical components to organizations' IT infrastructure for the last few decades. They are the most common way to store data due to their ease of use, the wide variety of solutions they can support, and the well-established best practices with which they are designed. Relational databases are useful for storing data elements that are related and must be stored in a consistent manner. This chapter will discuss the key features of a relational database, the different relational database offerings in Azure, basic management tasks for relational databases, and common query techniques for relational data.

## Relational Database Features

Relational databases store data as collections of entities in the form of tables. In the context of data, entities can be described as nouns, such as persons, companies, countries, or products. Tables contain structured data that describes an entity and are composed of zero or more rows and one or more columns of data. Some of the columns might be special columns that are used to uniquely identify each row or act as a reference to another table that they might be related to. Rows might not include values for each column, but because relational databases are designed with a rigid schema, the row will still include that column in its definition. Default or null values are used when a value is not provided for a row. This organized approach to data storage allows relationships between entities that can easily be queried by a data analyst or a data processing solution. Let's examine the features of a relational database, starting with design considerations.

### Relational Database Design Considerations

Design considerations for relational databases largely depend on what type of solution the database will be powering. As discussed in Chapter 1, "Core Data Concepts," relational databases are commonly used to power online transaction processing (OLTP) and analytical systems. Solutions that are powered by OLTP databases have different write and read requirements than that of an analytical database. Even though OLTP databases often serve as data sources for data warehouses or online analytical processing (OLAP) systems, these requirements make it necessary to distribute and store data differently in each system.

## OLTP Workload Design Considerations

Transactional data that is stored in an OLTP database involve interactions that are related to an organization's activities. These can include payments received from customers, payments made to suppliers, or orders that have been made. Typical OLTP databases are optimized to handle data that is written to them and must be able to ensure that transactions adhere to ACID properties (see Chapter 1 for more information on ACID properties). This will guarantee the integrity of the records that are stored. Relational database management systems (RDBMSs) typically enforce these rules using locks or row versioning.

Regardless of whether a transaction is reading, inserting, updating, or deleting data, the data involved in the transaction must be reliable. This becomes even more true as the number of users running transactions concurrently on the same pieces of data increases, resulting in the following issues:

- Dirty reads can occur when a transaction is reading data that is being modified by another transaction. The transaction performing the read is reading the modified data that has not yet been committed. This potentially results in an inaccurate result set if the transaction modifying the data is rolled back to the original values.
- Nonrepeatable reads occur when a transaction reads the same row several times and returns different data each time. This is the result of one or more other transactions being able to modify the data between the reads within the transaction.
- Phantom reads occur when two identical queries running in the same transaction return different results. This can happen when another query inserts some data in between the execution of the two queries, resulting in the second query returning the newly inserted data.

To mitigate these issues, a transaction will request locks on different types of resources, such as rows and tables, that the transaction is dependent on. Transaction locks prevent dirty, nonrepeatable, and phantom reads by blocking other transactions from performing modifications on data objects involved in the transaction. Transactions will free their locks from a resource once they have finished reading/modifying it. While locks are critical for ensuring consistency, they can cause long wait times for users that have issued transactions that are being blocked. The following isolation levels can be assigned to a transaction to balance consistency versus performance depending on its requirements:

- *Read Uncommitted* is the lowest isolation level, only guaranteeing that physically corrupt data is not read. Transactions using this isolation level run the risk of returning dirty reads since uncommitted data is read.
- *Read Committed* transactions issue locks on involved data at the time of data modification to prevent other transactions from reading dirty data. However, data can be modified by other transactions, which can result in nonrepeatable or phantom reads. This is the default isolation level for SQL Server and Azure SQL Database.

- *Repeatable Read* transactions issue read and write locks on involved data until the end of the transaction. No other transaction can modify data involved by a repeatable read transaction until the transaction has completed. However, other transactions can insert new rows into tables involved in a repeatable read transaction. This could possibly result in phantom reads occurring.
- *Serializable* is the highest isolation level and completely isolates transactions from one another. Statements cannot read data that has not yet been committed by a transaction with serializable isolation. What's more is that statements cannot modify data that is being read by a transaction whose isolation is set to serializable.

SQL Server and Azure SQL Database also allow users to use row versioning to maintain versions of rows that are modified. Transactions can be specified to use row versions to view data as it existed at the start of the transaction instead of protecting it with locks. This allows the transaction to read a consistent copy of the data while mitigating performance concerns from locking. The following isolation levels support row versioning:

- *Read Committed Snapshot* is a version of the Read Committed isolation level that uses row versioning to present each statement in the transaction with a consistent snapshot of the data as it existed at the beginning of the statement. Locks are not used to protect the data from updates by other transactions. To enable Read Committed Snapshot, set the READ_COMMITTED_SNAPSHOT database option to ON.
- *Snapshot* isolation uses row versioning to return rows as they existed at the start of the transaction, regardless of whether another transaction modifies those rows. To enable Snapshot isolation, set the ALLOW_SNAPSHOT_ISOLATION database option to ON.

NOTE

Since each version is stored in `tempdb`, special maintenance considerations must be taken into consideration. Please refer to the following link for more information on isolation levels based on row versioning: `https://docs.microsoft.com/en-us/sql/relational-databases/sql-server-transaction-locking-and-row-versioning-guide?view=sql-server-ver15#Row_versioning`.

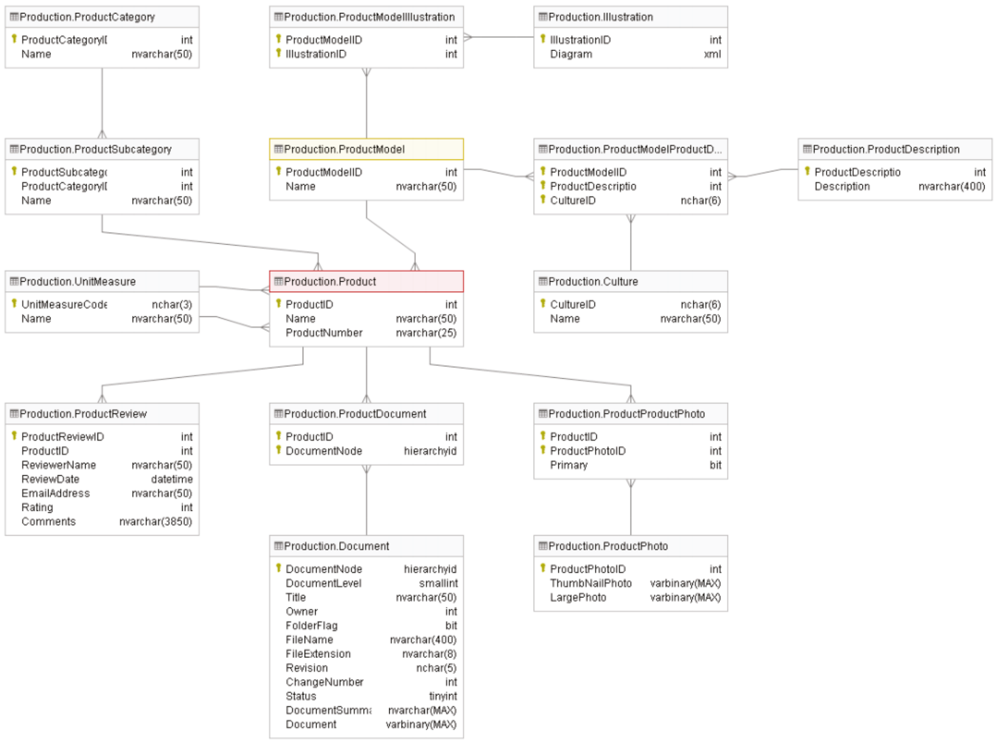
Maintaining ACID compliancy while also ensuring a premium performance experience is no easy task. Design best practices for OLTP databases are able to accomplish this by breaking up data into smaller chunks that are less redundant, also known as *data normalization*. There are a few rules for data normalization, which can be defined as follows:

- First normal form (1NF) involves eliminating repeating groups in individual tables, creating separate tables for each set of related data, and identifying each set of related data with a primary key. This is essentially stating that you should not use multiple fields in a single table to store similar data. For example, a retail company may have customers that make multiple orders at different periods of time. Instead of duplicating the customers' information each time they place an order, place all customer information in a separate table called Customers and identify each customer with a unique primary key.

- Second normal form (2NF) takes 1NF a step further. Along with the rules that define 1NF, 2NF also involves creating separate tables for sets of values that apply to multiple records and then relating those tables via a foreign key. For example, if a customer's address is needed by the Customer, Order, and Shipping tables, separate the addresses into a single table such as the Customer table or their own Address table.
- Third normal form (3NF) builds on 1NF and 2NF by including a requirement to eliminate fields in tables that do not depend on the key. For example, let's assume that each product being sold by the retail company includes several subcategories. If each subcategory is stored in the Products table, then each product will be duplicated numerous times to include each subcategory. In this case, it is more efficient to create a Product Subcategory table and relate it to the Products table.

The more normalized a database is, the more efficiently the database will handle write operations. This is because normalized data avoids extra processing for redundant data. Typical OLTP databases follow 3NF to ensure that database writes are as efficient as possible. Figure 2.1 is a partial example of the AdventureWorks OLTP entity relationship diagram (ERD), focusing on entities that are related to products manufactured and sold by AdventureWorks. The entire ERD can be found at `https://dataedo.com/download/AdventureWorks.pdf`.

**FIGURE 2.1** OLTP ERD



In this diagram, every entity that has multiple records is broken up into multiple tables to avoid redundant data storage. Entities that are related to one another can be related one or many times. For example, each product category has multiple product subcategories. Therefore, the relationship between the ProductCategory table and the ProductSubcategory table is one-to-many. The *crow's feet* shape in the relationship signals that there are many product subcategories for each product category.

While this level of data normalization is highly efficient for writing and storing individual transactions, it can be less efficient for applications that perform large numbers of read operations. Queries that are issued from read-heavy applications (e.g., reporting and analytical applications) potentially require many joins to de-normalize the data, making these queries very long and complex. Read operations that perform aggregations over large amounts of data are also very resource intensive for OLTP databases and can cause blocking issues for other transactions issued against the database. It is for these reasons that analytical databases carry different design best practices than their OLTP counterparts.

## Analytical Workload Design Considerations

Data warehouses and online analytical processing (OLAP) systems are optimally designed for read-heavy applications. While OLTP systems focus on storing current transactions, data warehouses and OLAP models focus on storing historical data that can be used to measure a business's performance and predict what future actions it should take.

Data warehouses serve as central repositories of data from one or more disparate data sources, including various OLTP systems. Not only does this eliminate the burden of running analytical workloads from the OLTP database, it also enriches the OLTP data with other data sources that provide useful information for decision makers. Data warehouses can store data that is processed in batch and in real time to provide a *single source of truth* for an organization's analytical needs. Data analysts commonly run analytical queries against data warehouses that return aggregated calculations that can be used to support business decisions.

Data warehouses can be built using one of the SMP database offerings on Azure, such as Azure SQL Database, or on the MPP data warehouse Azure Synapse Analytics dedicated SQL pools. The choice largely depends on the amount of historical data that is going to be stored and the nature of the queries that will be issued to the data warehouse. A good rule of thumb is that if the size of the data warehouse is going to be less than 1 terabyte, then Azure SQL Database will do the trick. However, this is a general statement, and more consideration is needed when deciding between SMP or MPP. Chapter 5, "Modern Data Warehouses in Azure," covers more detail on what to consider when designing a modern data warehouse.

OLAP models extract commonly used data for reporting from data warehouses to simplify data analysis. Like data warehouses, OLAP models are used for read-heavy scenarios and typically include the following predefined features to allow users to see consistent results without having to write their own logic:

- Aggregations that can be immediately reported against
- Time-oriented calculations

OLAP models come in two flavors: multidimensional and tabular. Multidimensional cubes such as those created with SQL Server Analysis Services (SSAS) were used in traditional business intelligence (BI) solutions to serve data as dimensions and measures. Tabular models such as Azure Analysis Services and models built in Power BI serve data using relational modeling constructs (e.g., tables and columns) while storing metadata as multidimensional modeling constructs (e.g., dimensions and measures) behind the scenes. Tabular models have become the standard for OLAP models as they use similar design patterns to relational databases, make use of columnar storage that optimally compresses data for analytics and leverages an easy-to-learn language (DAX) that data analysts can use to create custom metrics. Chapter 6, "Reporting with Power BI," will describe in detail tabular models and how they are used in Power BI.

Data warehouses and OLAP models store data in a way that is designed to be easy for analysts and developers to read. Tables in analytical systems are defined to be easily understood by business users so that they do not have to rely on IT every time they need to produce new analysis against historical data. Instead of using strict nomenclature and normalized rules that make OLTP systems ideal for storing transactional data, analytical systems flatten data so that business users can easily query data without having to join several tables together.

One common design pattern for data warehouses and OLAP models is the *star schema*. Star schemas denormalize data taken from OLTP systems, resulting in some attributes being duplicated in tables. This is done to make the data easier for analysts to read, allowing them to avoid having to join several tables in their queries. While de-normalization is not optimal for write-heavy, transactional workloads, it will increase the performance of read-oriented, analytical workloads.

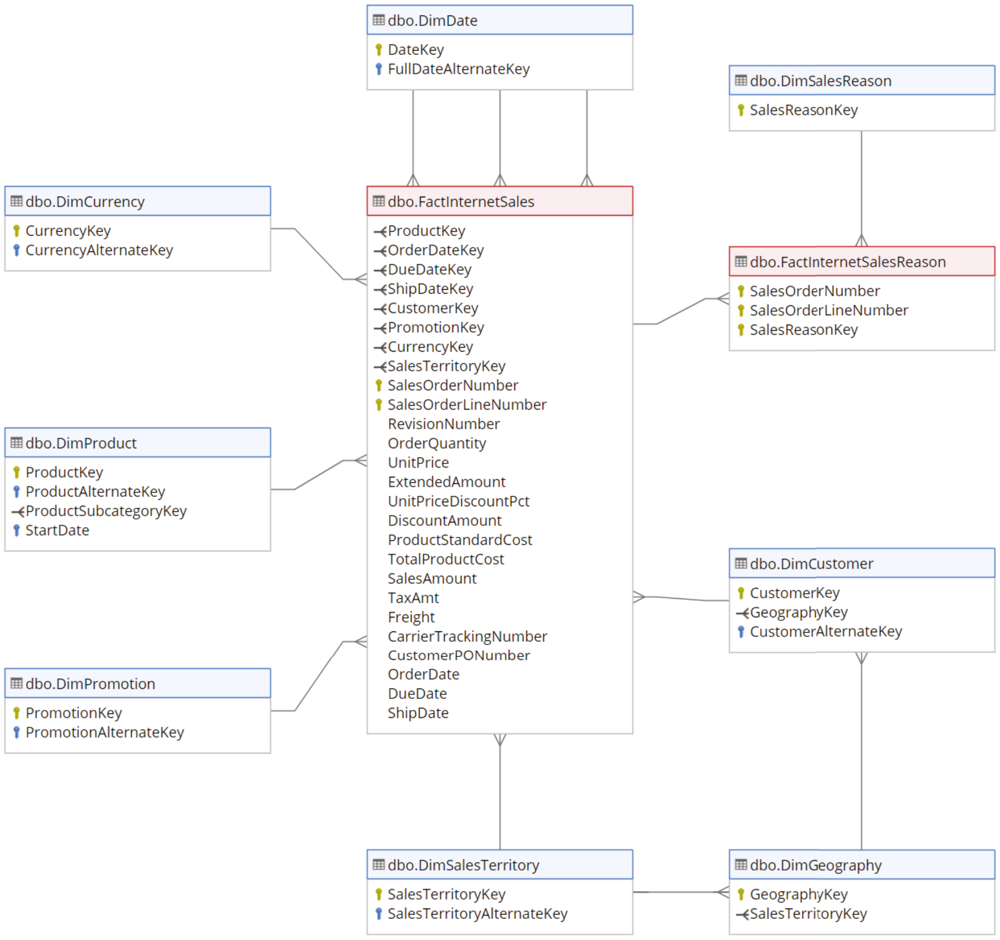
Star schemas work by relating business entities, also known as the nouns of the business, to measurable events. These can be broken down into the following classifications that are specific to a star schema:

- *Dimension tables* store information about business entities. Dimension tables store descriptive columns for each entity and a key column that serves as a unique identifier. Examples include date, customer, geography, and product dimensions. Dimension tables typically store a relatively small number of rows but many columns, depending on how many descriptors are necessary for a given dimension.
- *Fact tables* store measurable observations or events such as Internet sales, inventory, or sales quotas. Along with numeric measurements, fact tables contain dimension key columns for each dimension that a measure or observation is related to. These relationships determine the granularity of the data in the fact table. For example, an Internet sales fact table that has a dimension key for date is only as granular as the level of detail stored in the date dimension table. If the date dimension table only includes details for years and months, then queries performing time-based calculations will only be able to drill down to monthly sales. However, if it includes details for years, quarters, months, weeks, days, and hours, then queries will be able to perform more fine-grained analysis of the data.

Figure 2.2 is a partial example of the AdventureWorks DW star schema, focusing on dimensions and facts related to Internet sales for products manufactured and sold by

AdventureWorks. The entire diagram can be found at `https://dataedo.com/samples/html/Data_warehouse/doc/AdventureWorksDW_4/modules/Internet_Sales_101/module.html`.

**FIGURE 2.2** Star schema



This diagram shows the relationship between the *nouns* involved in an online sale and the associated metrics. While not illustrated in the image, if you go to the link in the preceding paragraph, you will find more details on each dimension table and will see that they have many columns that provide high granularity for the sales metrics.

OLAP models take star schemas a step further by including business logic and predefined calculations that are ready to be used in reports. This level of abstraction that allows users to focus on building business-critical reports without needing to write SQL queries that perform aggregations and joins over the underlying data is known as a *semantic layer*. Semantic layers are typically placed over data pulled from a data warehouse. Along with the business-friendly names that come with a star schema, semantic layers store calculations that allow users to easily filter and summarize data.

## Relational Data Structures

Relational databases are composed of several different components. Take an OLTP database that powers a retail company's POS for example. This database probably has a customer table that contains rows for every customer that has made a purchase. The table can include columns for each customer's first name, last name, phone number, address, and more. Every column has a predefined data type that inserted values must adhere to. If a customer chooses not to give a piece of information such as their phone number, a null value can be added as a placeholder so that the row maintains the structure of the table's schema. Every row is also assigned an ID that uniquely identifies the customer, also known as a *primary key*. Some columns, such as the ID column, are also used to relate to other tables such as one that stores more information about the products involved in a purchase. This is known as a *foreign key*. The customer table can also include *indexes* that optimize how the data is organized so that queries can quickly retrieve data. These database structures and others are defined in the following sections.

### Tables

Tables are structured database objects that store all the data in a database. Data is organized into rows and columns, with rows representing records of data and columns representing a field in the record. Along with user-defined tables that persist data, users can choose to create temporary tables that briefly store data that does not need to be persisted long term. These come in two varieties:

- Local temporary tables are only visible to the instance of a user connection, also known as a session, that they are built in. They are deleted as soon as the session is disconnected.
- Global temporary tables are visible to any user after they are created and are deleted when all user sessions referencing the table are disconnected.

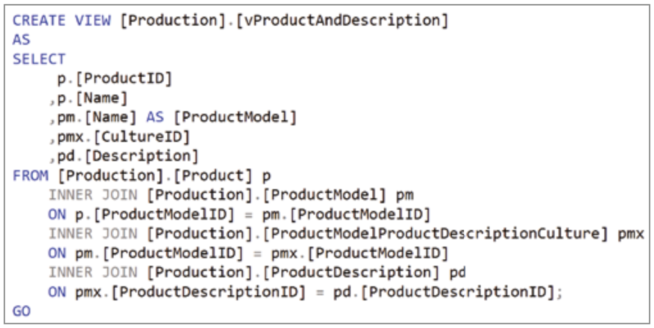
SMP and MPP databases allow users to create partitions on tables to horizontally distribute data across multiple filegroups in a database. This makes large tables easier to manage by allowing users to access individual partitions of data quickly and efficiently while the integrity of the overall table is maintained. MPP systems such as Azure Synapse Analytics dedicated SQL pools take this a step further. Along with being able to partition data across filegroups, MPP systems spread data across multiple *distributions* on one or more compute nodes. The types of distributed tables available in Azure Synapse Analytics dedicated SQL pools and when to use each are covered in Chapter 5, "Modern Data Warehouses in Azure."

### Views

*Views* are virtual tables whose contents are defined by a query. The rows and columns of data in a view come from tables referenced in the query that define the view. They act as a virtual layer to filter and combine data from regularly queried tables. Users can simplify their queries since views handle the complex filtering and joining of data that would normally need to be handled by the user. They are also useful security mechanisms as users do not need permission to the underlying tables that make up the views.

Figure 2.3 is an example of a view definition taken from the AdventureWorks OLTP database. This view queries the ProductModel, ProductModelProductDescriptionCulture, and ProductDescription tables to compile a list of products sold and their descriptions in multiple languages.

**FIGURE 2.3** View definition



```
CREATE VIEW [Production].[vProductAndDescription]
AS
SELECT
     p.[ProductID]
    ,p.[Name]
    ,pm.[Name] AS [ProductModel]
    ,pmx.[CultureID]
    ,pd.[Description]
FROM [Production].[Product] p
    INNER JOIN [Production].[ProductModel] pm
    ON p.[ProductModelID] = pm.[ProductModelID]
    INNER JOIN [Production].[ProductModelProductDescriptionCulture] pmx
    ON pm.[ProductModelID] = pmx.[ProductModelID]
    INNER JOIN [Production].[ProductDescription] pd
    ON pmx.[ProductDescriptionID] = pd.[ProductDescriptionID];
GO
```

This view allows users querying product description information to simplify their queries from performing joins on multiple tables to only reading from one database object.

A special type of view that can be used to improve the performance of complex analytical queries that are issued against large data warehouse datasets are *materialized views*. Unlike regular views that are generated each time the view is used, materialized views are preprocessed and stored in the data warehouse. The data stored in a materialized view is updated as it is updated in the underlying tables. Materialized views that are defined by complex analytical queries improve performance and reduce the amount of time required to prepare data for analysis by pre-aggregating data and storing it in a manner that is ready to be used in reports.

NOTE

To create a materialized view in Azure Synapse Analytics dedicated SQL pools, you will need to issue a CREATE MATERIALIZED VIEW statement instead of a CREATE VIEW statement that you would with a normal view. Materialized views also require an explicit distribution type, like tables stored in dedicated SQL pools. More on distribution types in Chapter 5, “Modern Data Warehouses in Azure.”

## Indexes

Consider the index at the end of this book. Its purpose is to sort keywords and provide each keyword's location in the book. Database indexes work very similarly in that they sort a list of values and provide pointers to the physical locations of those values. Ideally, indexes are designed to optimize the way data is stored in database tables to best serve the types of queries that are issued to them.

Depending on the workload, indexes physically store data in a row-wise format (rowstore) or a column-wise format (columnstore). If queries are searching for values, also known as seeks, or for a small range of values, then rowstore indexes such as clustered and nonclustered indexes are ideal. On the other hand, columnstore indexes are best for database tables that store data that is commonly scanned and aggregated. The following are descriptions of the three commonly used types of indexes:

- *Clustered indexes* physically sort and store data based on their values. There can only be one clustered index because clustered indexes determine the physical order of the data. Columns that include mostly unique values are ideal candidates for clustered indexes. Clustered indexes are automatically created on primary key columns for this reason.
- *Nonclustered indexes* contain pointers to where data exists. There can be more than one nonclustered index on a database, and each one can be composed of multiple columns depending on the nature of the queries issued to the database. For example, queries that return data based on specific filter criteria can benefit from a nonclustered index on the columns being filtered. The nonclustered index allows the database engine to quickly find the data that matches the filter criteria.
- *Columnstore indexes* use column-based data storage to optimize the storage of data stored in a data warehouse. Instead of physically storing data in a row-wise format like that of a clustered or nonclustered index, columnstore indexes store data in a column-wise format. This provides a high level of compression and is optimal for analytical queries that perform aggregations over large amounts of data.

Proper index design can be the difference between a poorly performing database and one that runs like a charm. While index design best practices are out of scope for this book, I recommend the following article for guidelines on choosing an index strategy: `https://docs.microsoft.com/en-us/sql/relational-databases/sql-server-index-design-guide?view=sql-server-ver15`.

## Stored Procedures

Stored procedures are groups of one or more T-SQL statements that perform actions on data in a database. They can be executed manually or via an external application (e.g., custom .NET application, Azure Data Factory). They can also be scheduled to run at predetermined periods of time with a SQL Server Agent job, such as every hour or every night at midnight. Stored procedures can accept input parameters and return multiple values as output parameters to the application calling them.

Code that is frequently used to perform database operations is an ideal candidate to be encapsulated in stored procedures. This eliminates the need to rewrite the same code repeatedly, which also reduces the chances of errors from code inconsistency. The application tier is also simplified since applications will only need to execute the stored procedure instead of needing to maintain and run entire blocks of T-SQL code.

### Functions

Functions are like stored procedures in that they encapsulate commonly run code. The major difference between a user-defined function in SQL and a stored procedure is that functions must return a value. Stored procedures can be used to make changes to data without ever returning a response to the user running the stored procedure. Functions, on the other hand, can only return data that is typically the result of a complex calculation. Functions accept parameters and return values as either a single scalar value or a result set.

### Triggers

Triggers are T-SQL statements that are executed in response to a variety of events. These events can be DDL, DML, or login related. Triggers are typically used when you want to do the following:

- Prevent certain changes to columns in tables.
- Perform an action based on a change to database schemas or underlying data.
- Log changes to the database schema.
- Enforce relational integrity throughout the database.

## Relational Database Offerings in Azure

Until recently, most organizations hosted their database systems in on-premises datacenters that they owned or leased. They were responsible for applying updates to the database software and had to make sure that the hardware hosting the databases was properly maintained. Business continuity aspects such as database backup management, high availability (HA), and disaster recovery (DR) standards would need to be implemented to ensure minimal downtime in case of database corruption or server downtime. Scalability is also a concern, as database servers that outgrow compute allocated to them require someone to physically add compute to the server. All these items require additional hardware and levels of expertise from employees, thus increasing the total cost of ownership (TCO) for a database.

Cloud-based hosting has fundamentally shifted how organizations calculate TCO for their relational databases. Many operations that surround database upgrades or patching, business continuity, and scalability are handled by the cloud company. This allows organizations to shift their focus from maintaining hardware and managing business continuity concerns to being able to purely focus on the needs of the database users. Provisioning and scaling a database is also much easier as almost every requirement is preconfigured. Shortly put, databases can be easily deployed in Azure with the click of a button and scaled up and down with a slider (more on this later in the chapter).

Before getting into the different relational databases offerings in Azure, it's important to understand the three types of cloud computing services. Having a foundational knowledge of how each of these are implemented is paramount to understanding the responsibilities and the TCO for hosting a database on Azure.

- *Infrastructure as a Service (IaaS)* offerings in Azure provide customers with the ability to create virtual infrastructure that mirrors an on-premises environment. IaaS offerings give organizations the ability to easily migrate their on-premises infrastructure to similar IaaS-based offerings in Azure without needing to completely redesign their applications using a cloud native approach. This is a typical first step for moving to the cloud as it allows organizations to offload the management of their hardware to Microsoft using a *lift-and-shift* strategy. While IaaS deployments allow organizations to no longer worry about maintaining the hardware powering an application, they will still need to manage maintenance at the operating system (OS) and application level. IaaS offerings include virtual machines that host services that would typically be hosted in a customer's on-premises environment, such as SQL Server, and are connected via an Azure Virtual Network (VNet). These services can easily connect to an organization's existing network infrastructure, allowing them to utilize a hybrid cloud strategy.
- *Platform as a Service (PaaS)* takes IaaS a step further by abstracting the OS and application software from the user. When deploying a PaaS offering, organizations can specify the resources they would like deployed, an initial size and compute tier depending on the intensity of the workload, what Azure region they would like them deployed to, and other optional or service-specific requirements. Azure will then provision the necessary resources to meet those specific requirements. Once deployed, all OS and software maintenance such as business continuity, upgrades, and patches are handled by Azure. This allows organizations to minimize the amount of effort required to maintain these services and instead focus on using them to build solutions that impact the business. Like IaaS offerings, PaaS offerings can also be interconnected via a VNet and connected to an organization's existing on-premises network infrastructure. PaaS services include Azure SQL Database, Azure SQL Managed Instance (MI), and all the open-source relational database offerings that are hosted on Azure SQL.
- *Software as a Service (SaaS)* offerings represent the highest level of abstraction available to an organization hosting its infrastructure and applications on the cloud. Organizations simply purchase the number of licenses required for the service and then use it. Typical examples of SaaS offerings include Power BI Online and Office 365. None of the relational database offerings discussed in this chapter are SaaS offerings.

NOTE

IaaS, PaaS, and SaaS are critical components of the Microsoft Azure Fundamental AZ-900 exam. If you would like to learn more about the three types of cloud computing services, please read Jim Boyce's *Microsoft Certified Azure Fundamental Exam Guide* (Wiley, 2021). This book provides a fundamental knowledge of Azure and provides further in-depth information on IaaS, PaaS, and SaaS applications.

## Azure SQL

*Azure SQL* is a broad term used to describe the family of SMP relational database products in Azure that are built upon Microsoft's SQL Server engine. These include one IaaS option with SQL Server on Azure Virtual Machines (VM) and two PaaS options with Azure SQL MI and Azure SQL Database. Azure SQL Database can be broken down even further into two different options: single database and elastic pool. There are also several service tiers available for each offering that best suit different types of workloads. With so many options available, organizations must weigh several factors when deciding which Azure SQL option is the most appropriate for their use cases:

- *Cost*—All three options include a base price that covers underlying infrastructure and licensing. Each option also includes hybrid licensing benefits that allow organizations to apply on-premises SQL Server licenses to reduce the cost of the service. Keep in mind that hosting a database in a virtual machine will require additional administration overhead that the PaaS options don't require.
- *Service-level agreement (SLA)*—All three options provide high, industry-standard SLAs. PaaS options guarantee a 99.99 percent SLA, while IaaS guarantees a 99.95 percent SLA for infrastructure, meaning that organizations will need to implement additional mechanisms to ensure database availability. You can refer to the following documentation for more information regarding Azure SQL SLAs: `https://docs.microsoft.com/en-us/azure/azure-sql/azure-sql-iaas-vs-paas-what-is-overview#service-level-agreement-sla`.
- *Migration timeline*—This is a factor that must be considered if an organization is migrating to Azure as opposed to building an application from scratch with a cloud native design. Organizations may consider one option over the other depending on how long the timeline is. For example, databases can be migrated to a virtual machine in a relatively short amount of time because virtual machines can host the same version of SQL Server as an on-premises SQL Server instance. Azure SQL MI also provides nearly the same feature parity as an on-premises SQL Server, but some changes may need to be applied, especially if the database that will be migrated is hosted on an older version of SQL Server.
- *Administration*—Azure SQL Database and Azure SQL MI minimize overhead by managing typical database administration activities such as database backups, patches, version upgrades, HA, and threat protection. However, this also limits the range of custom administrative activities that can be performed.
- *Feature parity*—Because Azure SQL Database abstracts the OS and database server components from the user, there are certain features of SQL Server that are not supported in Azure SQL Database. These include cross-database joins, CLR integration, and SQL Server Agent. Azure SQL MI is nearly 100 percent feature compatible with SQL Server but still maintains a few differences such as features that rely on Windows-related objects. SQL Server on Azure VMs provides 100 percent feature parity because it is the same as a SQL Server instance hosted on an on-premises virtual machine. However,

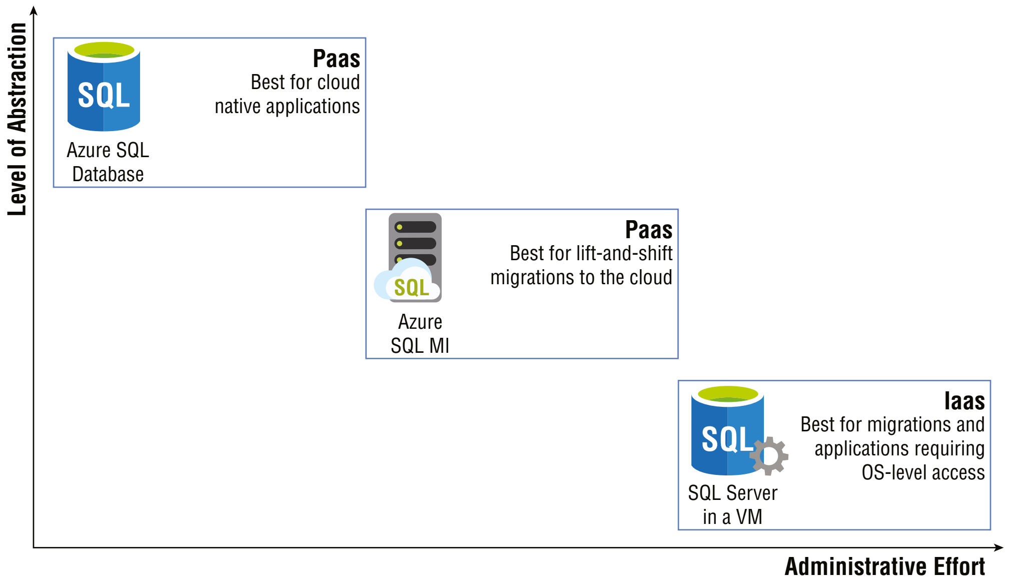
because of potential SQL Server version differences, there could be feature differences when migrating from an old version of SQL Server to a newer version of SQL Server on Azure VM or one of the PaaS options. Deprecated features and features that are incompatible with the PaaS options can be discovered using the Data Migration Assistant (DMA). The DMA will be covered in further detail later in this chapter.

NOTE

Most differences between SQL Server and Azure SQL Database are focused on server-related differences. Because Microsoft manages the backend server for an Azure SQL Database, features such as logins, server-level permissions, and EXECUTE AS LOGIN are not available. Instead, users can use database-scoped features such as database-level permissions and EXECUTE AS USER. More information on SQL Server and Azure SQL Database differences can be found at `https://docs.microsoft.com/en-us/azure/azure-sql/database/transact-sql-tsql-differences-sql-server`.

The various Azure SQL offerings come with different levels of abstraction and management. Figure 2.4 illustrates the relationship between abstraction and administrative effort for each option.

**FIGURE 2.4** Azure SQL abstraction vs. administrative effort


Level of Abstraction
SQL
Azure SQL Database
Paas
Best for cloud native applications
SQL
Azure SQL MI
Paas
Best for lift-and-shift migrations to the cloud
SQL
SQL Server in a VM
Iaas
Best for migrations and applications requiring OS-level access
Administrative Effort

As seen in the diagram, a SQL Server on Azure VM requires the most administrative effort because it provides full control over the SQL Server instance and the underlying OS. This is ideal for situations that require highly customized OS and/or database images or scenarios requiring very granular control over the SQL Server engine. Azure SQL MI removes the OS layer from the user's point of view but is more like an on-premises SQL Server instance than Azure SQL Database in that it provides a fully isolated environment encapsulated in a VNet and includes system databases. Users hosting their databases on Azure SQL MI still benefit from using a PaaS database in that patching, SQL Server version upgrades, backups, HA, DR, data encryption, auditing, and threat protection are all handled behind the scenes by Microsoft. Azure SQL Database completely abstracts the OS layer and database engine from users. Greenfield solutions that are developed using cloud native best practices typically use Azure SQL Database as their backend relational database.

Ultimately, choosing the right Azure SQL option comes down to the solution requirements and how much control is needed over the OS and database engine. The following sections explore each option in further detail.

## SQL Server on Azure Virtual Machine

There are several reasons why an organization would want to migrate its applications to the cloud. Perhaps the most common reason is to offload the maintenance of hardware and networking equipment that it either owns or leases to a cloud provider. Expiring datacenter leases or aging hardware force many companies to rethink how they manage their IT infrastructure. While many organizations will work to modernize their applications to be cloud native, most of them still use legacy applications that rely on features of SQL Server that are not available in the PaaS offerings of Azure SQL. There could also be specific situations that require fine-grained control over the database engine and the OS that it sits on. For these reasons, organizations may decide to migrate their existing SQL Server footprint to Azure SQL's IaaS offering: SQL Server on Azure VMs.

As far as the database engine is concerned, a SQL Server on Azure VM is no different than a SQL Server instance hosted on a physical server in an on-premises environment. This allows developers and database administrators to acclimate quickly to working with SQL Server in Azure. Engineers deploying a SQL Server on Azure VM can choose one of three approaches for doing so:

- Choose one of the available SQL Server VM images from the Azure marketplace. These images allow you to easily deploy a specific version of SQL Server on the OS of your choosing.
- Install your own SQL Server license on a VM. Users with existing VMs in Azure can choose to install SQL Server with an existing license to save the need of deploying a new VM.
- Lift-and-shift existing VMs from an on-premises environment to Azure with Azure Migrate. Azure Migrate is a tool that can be used to assess and migrate on-premises infrastructure to Azure. VMs hosting SQL Server can be migrated to Azure using Azure Migrate without needing to deploy a new VM through the Azure Marketplace. More information on migrating VMs to Azure with Azure Migrate can be found at `https://docs.microsoft.com/en-us/azure/migrate/migrate-services-overview#azure-migrate-server-migration-tool`.

Taking advantage of the ready-made images available in the Azure Marketplace greatly reduces the amount of time needed to provision a SQL Server VM in Azure. There are two licensing types available for SQL Server VMs: pay-as-you-go and bring your own license (BYOL). Pay-as-you-go simplifies licensing costs by billing you for the per-minute usage of the instance. Table 2.1 outlines the available pay-as-you-go SQL Server images in Azure.

**TABLE 2.1** Available Pay-As-You-Go SQL Server images

| Version | Operating System | Edition |
|---|---|---|
| SQL Server 2019 | Windows Server 2019 | Enterprise, Standard, Web, Developer |
| SQL Server 2019 | Ubuntu 18.04 | Enterprise, Standard, Web, Developer |
| SQL Server 2019 | Red Hat Enterprise Linux (RHEL) 8 | Enterprise, Standard, Web, Developer |
| SQL Server 2019 | SUSE Linux Enterprise Server (SLES) v12 SP5 | Enterprise, Standard, Web, Developer |
| SQL Server 2017 | Windows Server 2016 | Enterprise, Standard, Web, Express, Developer |
| SQL Server 2017 | Red Hat Enterprise Linux (RHEL) 7.4 | Enterprise, Standard, Web, Express, Developer |
| SQL Server 2017 | SUSE Linux Enterprise Server (SLES) v12 SP2 | Enterprise, Standard, Web, Express, Developer |
| SQL Server 2017 | Ubuntu 16.04 LTS | Enterprise, Standard, Web, Express, Developer |
| SQL Server 2016 SP2 | Windows Server 2016 | Enterprise, Standard, Web, Express, Developer |
| SQL Server 2014 SP2 | Windows Server 2012 R2 | Enterprise, Standard, Web, Express |
| SQL Server 2012 SP4 | Windows Server 2012 R2 | Enterprise, Standard, Web, Express |
| SQL Server 2008 R2 SP4 | Windows Server 2008 R2 | Enterprise, Standard, Web, Express |

Organizations who have already purchased SQL Server licenses can also apply those licenses to reduce the VM's SQL Server cost component. This is known as bring your own license, or BYOL for short. Table 2.2 outlines the available BYOL SQL Server images in Azure.

**TABLE 2.2** Available bring your own license SQL Server images

| Version | Operating System | Edition |
|---|---|---|
| SQL Server 2019 | Windows Server 2019 | Enterprise BYOL, Standard BYOL |
| SQL Server 2017 | Windows Server 2016 | Enterprise BYOL, Standard BYOL |
| SQL Server 2016 SP2 | Windows Server 2016 | Enterprise BYOL, Standard BYOL |
| SQL Server 2014 SP2 | Windows Server 2012 R2 | Enterprise BYOL, Standard BYOL |
| SQL Server 2012 SP4 | Windows Server 2012 R2 | Enterprise BYOL, Standard BYOL |

NOTE

You can deploy an older version of SQL Server that is not available in the Azure Marketplace with PowerShell. To view available images, run the following command in a PowerShell window:

```
Import-Module -Name Az
$Location = <Azure Region the SQL Server VM will be deployed to>
Get-AzVMImageOffer -Location $Location -Publisher`
'MicrosoftSQLServer'
```

The available pay-as-you-go and BYOL SQL Server images are liable to change as new versions of SQL Server are introduced and older versions are deprecated. You can stay up to date on the available SQL Server VM images by referring to the tables in the following link: `https://docs.microsoft.com/en-us/azure/azure-sql/virtual-machines/windows/sql-server-on-azure-vm-iaas-what-is-overview#get-started-with-sql-server-vms`.

VM size and storage configuration must also be considered when creating a SQL Server Azure VM. There are multiple VM sizes available that include different virtual CPU quantities, memory sizes, and different disk sizes. Additional disks can be added to the VM depending on what is hosted in addition to SQL Server. There are also different categories of VM sizes that provide different baselines for performance, including these:

- *Memory optimized*—These provide stronger memory-to-vCPU ratios and are the Microsoft-recommended choice for SQL Server VMs on Azure.

- *General purpose*—These provide balanced memory-to-vCPU ratios and best serve smaller workloads such as development and test, web servers, and smaller database servers.
- *Storage optimized*—These are designed with optimized disk throughput and input-output (I/O) and are strong options for data analytics workloads.

These are general recommendations and should be used with application performance metrics to make the most appropriate VM choice for different workloads. Keep in mind that VMs use a pay-as-you-go cost model and can be stopped when not needed so that you are not charged during those times. However, most SQL Server VMs will need to stay online unless the SQL Server instance is a test instance. Organizations that will be using one or more SQL Server VMs for one or three years can purchase *Azure Reserved Virtual Machine Instances*. Once applied to a VM, Azure Reserved Virtual Machine Instances discount the cost of the virtual machine and compute costs.

NOTE

It is important to determine the right VM size before purchasing a reservation. The following link provides more information on Azure Reserved Virtual Machine Instances and determining the right VM size: `https://docs.microsoft.com/en-in/azure/virtual-machines/prepay-reserved-vm-instances?toc=/azure/cost-management-billing/reservations/toc.json#determine-the-right-vm-size-before-you-buy`.

Deploying a ready-made SQL Server VM image from the Azure Marketplace will include a default storage configuration for data, log, and tempdb files. While these configurations are optimal for general workloads, many workloads may benefit from different ones. There may also be a need to optimize for cost versus performance for non-production workloads. Regardless of workload type, these are some general checklist items that should be considered when configuring storage for a SQL Server VM on Azure:

- Place data, log, and tempdb files on separate drives.
- Place tempdb on the local SSD drive. This drive is ephemeral and will deallocate resources when the VM is stopped.
- Consider using standard HDD/SDD storage for development and test workloads.
- Use premium SSD disks for data and log files for production SQL Server workloads.
    - Use P30 and/or P40 disks for data files to ensure caching support.
    - Use P30 through P80 disks for log files.

Collecting storage performance metrics for workloads that will be migrated to Azure will help determine the most appropriate disk configuration. More information on SQL Server on Azure VM storage configurations can be found at `https://docs.microsoft.com/en-us/azure/azure-sql/virtual-machines/windows/performance-guidelines-best-practices-storage`.

## Business Continuity

There are multiple solutions available in Azure to ensure that data hosted on SQL Server VMs is highly available in the event of several outage scenarios, ranging from planned downtime to datacenter-level disasters. These include solutions that provide database backup management at the database level and high availability and disaster recovery (HADR) capabilities at both the VM and database levels.

Azure provides business continuity for disk storage by creating copies of the data stored on disk and storing them on Azure Blob storage. This type of redundancy can be broken down with the following options:

- *Locally redundant storage (LRS)* creates three copies of the data stored on disk and stores them in the same location in the same Azure region.
- *Geo-redundant storage (GRS)* stores three copies of the disk data in the same Azure region as the VM and then stores an additional three copies in a separate region.

While these services provide redundancy for data stored on Azure VMs, they should not be relied on as the only business continuity solution for SQL Server data. Database backups should also be taken to protect against application or user errors. Also, GRS does not support the data and log files to be stored on separate disks. Data from these two files is copied independently and asynchronously, creating a risk of losing data in the event of an outage.

Organizations can choose to set up their own database backup strategy through maintenance plans that are run as a SQL Server Agent job on a scheduled basis. Backups can be stored on local storage or in Azure Blob storage. Azure also allows organizations to offload this process by using a service called *Automated Backup*. This service regularly creates database backups and stores them on Azure Blob storage without requiring a database administrator to set up the job on the database engine.

For true database-level HADR, organizations can add databases hosted on SQL Server VMs to a *SQL Server Always On availability group*. Availability groups, or AGs for short, replicate data from a set of user databases to one or more secondary SQL Server instances that are hosted on different VMs. The VMs, or server nodes, that host the primary and secondary SQL Server instances are clustered at the OS level. The cluster monitors the health of the server nodes and will promote a secondary server node to the primary if the existing primary experiences a failure.

Typical AG configurations include at least one secondary node in the same region as the primary to maintain HA and at least one secondary node in a different region for DR. Database connections will move, or failover, to the HA node during planned downtime for the primary node. If the primary node and the secondary nodes in the same region as the primary are down at the same time, database connections will failover to the DR node in the other region.

AG configurations are not limited to Azure-only VMs. Hybrid scenarios are possible, allowing organizations to add on-premises SQL Server instances to the solution. This requires VPN connectivity between the Azure network that SQL Server Azure VM is in and the on-premises network that the on-premises SQL Server is in. Network requirements for SQL Server VMs on Azure and hybrid scenarios will be discussed in the next section.

NOTE While these are common solutions used to create business continuity solutions for SQL Server on Azure VMs, there are more specific patterns designed to serve different scenarios. Refer to the following link to learn more about setting up HADR solutions for SQL Server on Azure VMs: `https://docs.microsoft.com/en-us/azure/azure-sql/virtual-machines/windows/business-continuity-high-availability-disaster-recovery-hadr-overview`.

### Network Isolation

A critical component of any IaaS offering is its ability to be completely self-isolated within a virtual network. Virtual networks in Azure, otherwise known as VNets, provide the backbone for isolating communication between different services. A VNet can include one or more subnets depending on the services that it is hosting. VNets can connect to other Azure VNets using a service called VNet peering as well as connect to on-premises networks through a point-to-site VPN, site-to-site VPN, or an Azure ExpressRoute. Hybrid connections are critical for organizations that have a presence in Azure and continue to maintain some of their applications in their on-premises environment.

VNets enable organizations to block specific IP address ranges and network protocols from being able to access resources connected to them. This includes blocking access to and from the public Internet. Databases hosted on SQL Server VMs on Azure are therefore restricted to only being able to communicate with applications that have been approved by an organization's network security team.

### Deploying through the Azure Portal

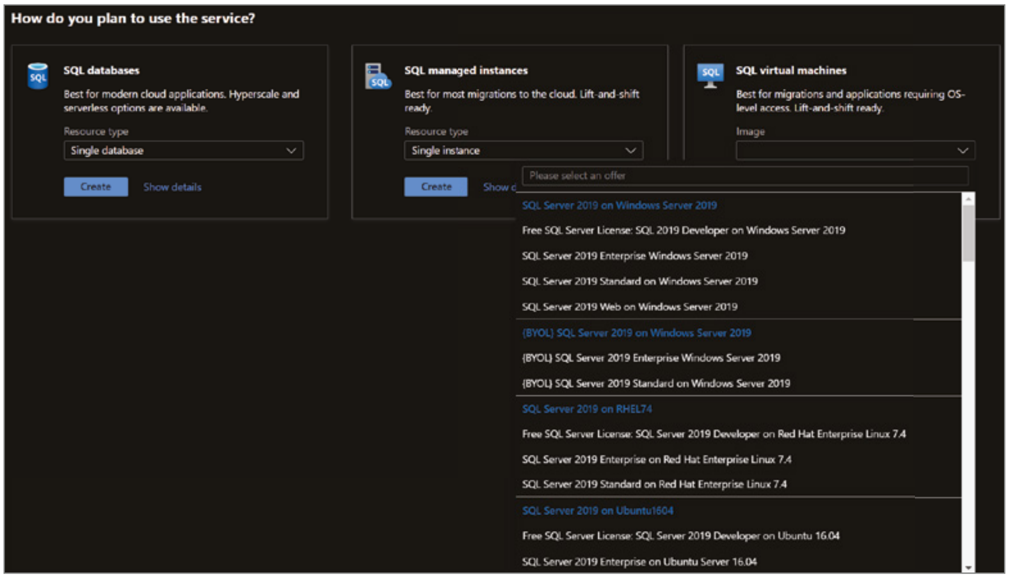
Deploying services in Azure can be done manually on the Azure Portal or automated using a scripting language (e.g., PowerShell or Bash) or an *Infrastructure as Code* template. SQL Server on Azure VMs are no different than any other service in this aspect, providing users multiple options for managing the deployment of their SQL Server databases on Azure. This section will cover the steps on how to manually deploy a SQL Server Azure VM through the Azure Portal. See the section "Deployment Scripting and Automation" later in this chapter to learn more about scripting and automating the deployment process for relational databases in Azure.

Use the following steps to create a SQL Server on Azure VM using the Azure Portal:

1. Log into `portal.azure.com` and search for *SQL virtual machines* in the search bar at the top of the page. Click *SQL virtual machines* to go to the SQL virtual machines page in the Azure Portal.
2. Click *Create* to start choosing the configuration options for your SQL Server on Azure VM.

3. Navigate to the SQL virtual machines option on the *Select SQL deployment option* page and select the VM image you would like to deploy. Figure 2.5 shows how this page is displayed and some of the options available after you click the *Image* drop-down arrow. Once you have selected an image, click the *Create* button to continue configuring the VM.

**FIGURE 2.5** Select a SQL virtual machine image.


How do you plan to use the service?

SQL databases
Best for modern cloud applications. Hyperscale and serverless options are available.
Resource type: Single database

SQL managed instances
Best for most migrations to the cloud. Lift-and-shift ready.
Resource type: Single instance

SQL virtual machines
Best for migrations and applications requiring OS-level access. Lift-and-shift ready.
Image

Create
Show details
Create
Show d
Please select an offer
SQL Server 2019 on Windows Server 2019
Free SQL Server License: SQL 2019 Developer on Windows Server 2019
SQL Server 2019 Enterprise Windows Server 2019
SQL Server 2019 Standard on Windows Server 2019
SQL Server 2019 Web on Windows Server 2019
(BYOL) SQL Server 2019 on Windows Server 2019
(BYOL) SQL Server 2019 Enterprise Windows Server 2019
(BYOL) SQL Server 2019 Standard on Windows Server 2019
SQL Server 2019 on RHEL74
Free SQL Server License: SQL Server 2019 Developer on Red Hat Enterprise Linux 7.4
SQL Server 2019 Enterprise on Red Hat Enterprise Linux 7.4
SQL Server 2019 Standard on Red Hat Enterprise Linux 7.4
SQL Server 2019 on Ubuntu1604
Free SQL Server License: SQL Server 2019 Developer on Ubuntu 16.04
SQL Server 2019 Enterprise on Ubuntu Server 16.04

4. The *Create a virtual machine* page includes eight tabs with different configuration options to tailor the SQL Server VM to fit your needs. Let's start by exploring the options available in the Basics tab. Along with the following list that describes each option, you can view a completed example of this tab in Figure 2.6.

   a. Choose the subscription and resource group that will contain the SQL Server VM. You can create a new resource group on this page if have not already created one.

   b. Enter a name for the VM.

   c. Choose the Azure region you wish to deploy the image to.

   d. Select whether you would like to enable high availability for the VM by using an Availability Zone or an Availability Set. Note that this is high availability for the VM and not for the SQL Server instance.

   e. Review the VM image selected and change it if necessary.

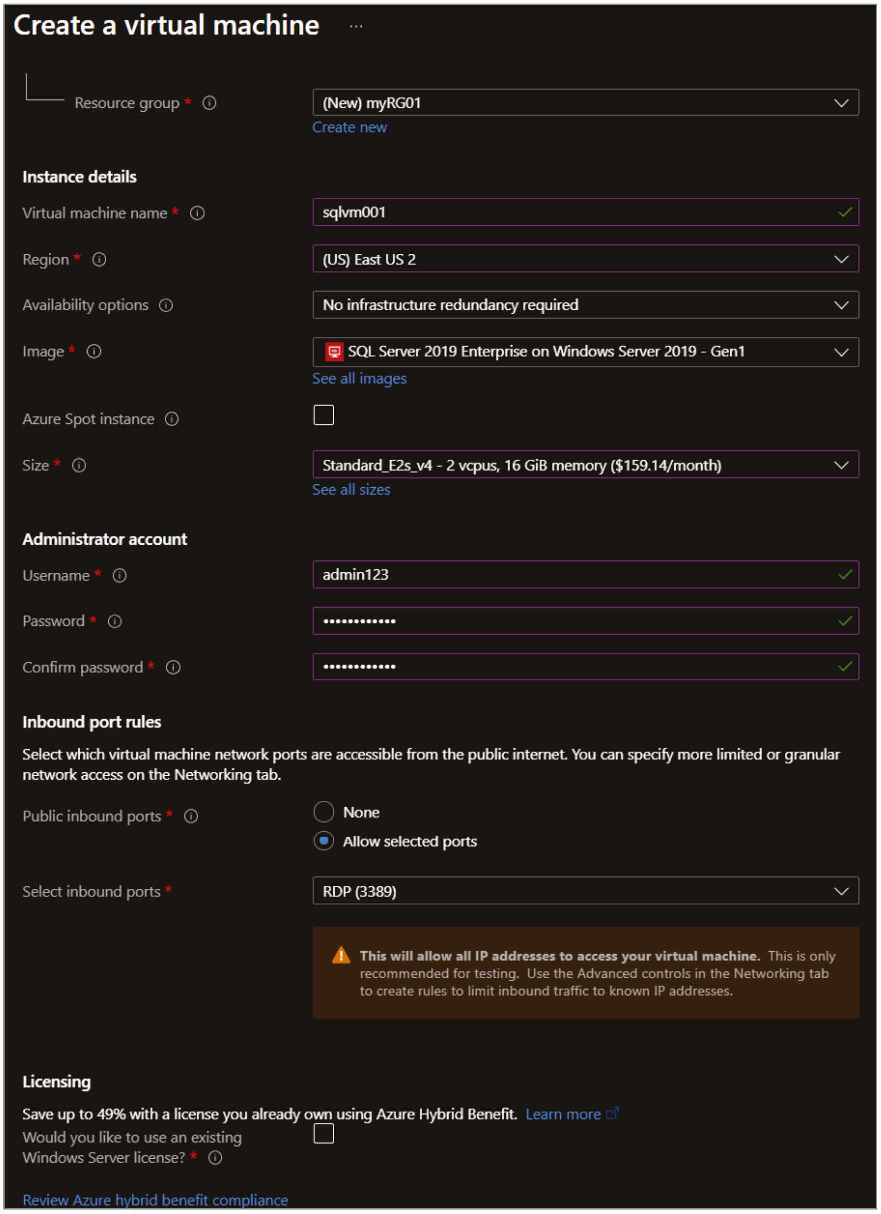
   f. Choose the VM size.

   g. Set a username and password for the administrator account.

h. Set any inbound network ports that you wish to be accessible from the public Internet.

i. The last optional step on this page is whether you would like to apply an existing Windows Server license to the VM to reduce its cost.

**FIGURE 2.6** Create a SQL virtual machine: Basics tab.



Create a virtual machine ···

Resource group * (New) myRG01
Create new

Instance details

| | |
|---|---|
| Virtual machine name * | sqlvm001 ✓ |
| Region * | (US) East US 2 |
| Availability options | No infrastructure redundancy required |
| Image * | SQL Server 2019 Enterprise on Windows Server 2019 - Gen1 |
| | See all images |
| Azure Spot instance | ☐ |
| Size * | Standard_E2s_v4 - 2 vcpus, 16 GiB memory ($159.14/month) |
| | See all sizes |

Administrator account

| | |
|---|---|
| Username * | admin123 ✓ |
| Password * | ••••••••••••• ✓ |
| Confirm password * | ••••••••••••• ✓ |

Inbound port rules

Select which virtual machine network ports are accessible from the public internet. You can specify more limited or granular network access on the Networking tab.

| | |
|---|---|
| Public inbound ports * | None [✓ Allow selected ports] |
| Select inbound ports * | RDP (3389) |

**This will allow all IP addresses to access your virtual machine.** This is only recommended for testing. Use the Advanced controls in the Networking tab to create rules to limit inbound traffic to known IP addresses.

Licensing

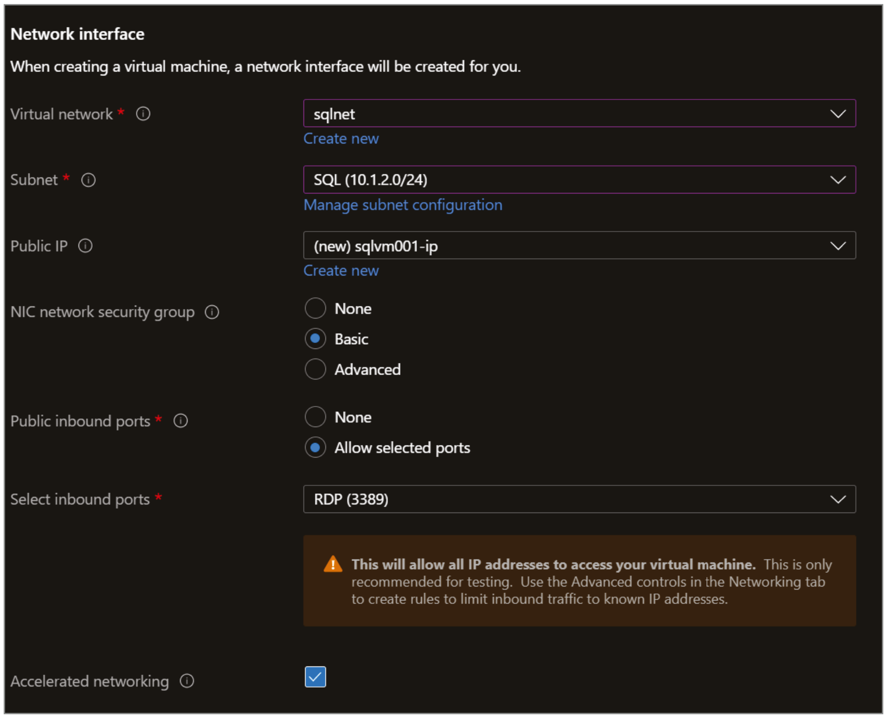
Save up to 49% with a license you already own using Azure Hybrid Benefit. Learn more

Would you like to use an existing Windows Server license? * ☐

Review Azure hybrid benefit compliance

5. The Disks tab focuses on the disk configuration for the OS. You can choose to change this from a Premium SSD to another disk type as well as change the encryption type used for the disk.
6. The Networking tab provides the following network configuration options for the VM. A completed example of this tab can be seen in Figure 2.7.
   a. Choose the virtual network that the VM will be located in.
   b. Choose a subnet within that virtual network for the VM.
   c. Optionally choose a public IP address to be used for communication outside of the virtual network.
   d. If needed, revise the open inbound ports selected in the Basics tab.

**FIGURE 2.7** Create a SQL virtual machine: Networking tab.



**Network interface**

When creating a virtual machine, a network interface will be created for you.

Virtual network * : sqlnet
Create new

Subnet * : SQL (10.1.2.0/24)
Manage subnet configuration

Public IP : (new) sqlvm001-ip
Create new

NIC network security group : None [✓ Basic] Advanced

Public inbound ports * : None [✓ Allow selected ports]

Select inbound ports * : RDP (3389)

**This will allow all IP addresses to access your virtual machine.** This is only recommended for testing. Use the Advanced controls in the Networking tab to create rules to limit inbound traffic to known IP addresses.

Accelerated networking : ✓

7. The Management tab allows you to customize features such as Azure Security Center monitoring, enabling automatic shutdown for the VM, and when OS patches should be applied.
8. The Advanced tab allows you to add any extensions or scripts to further customize the VM as it is being provisioned.

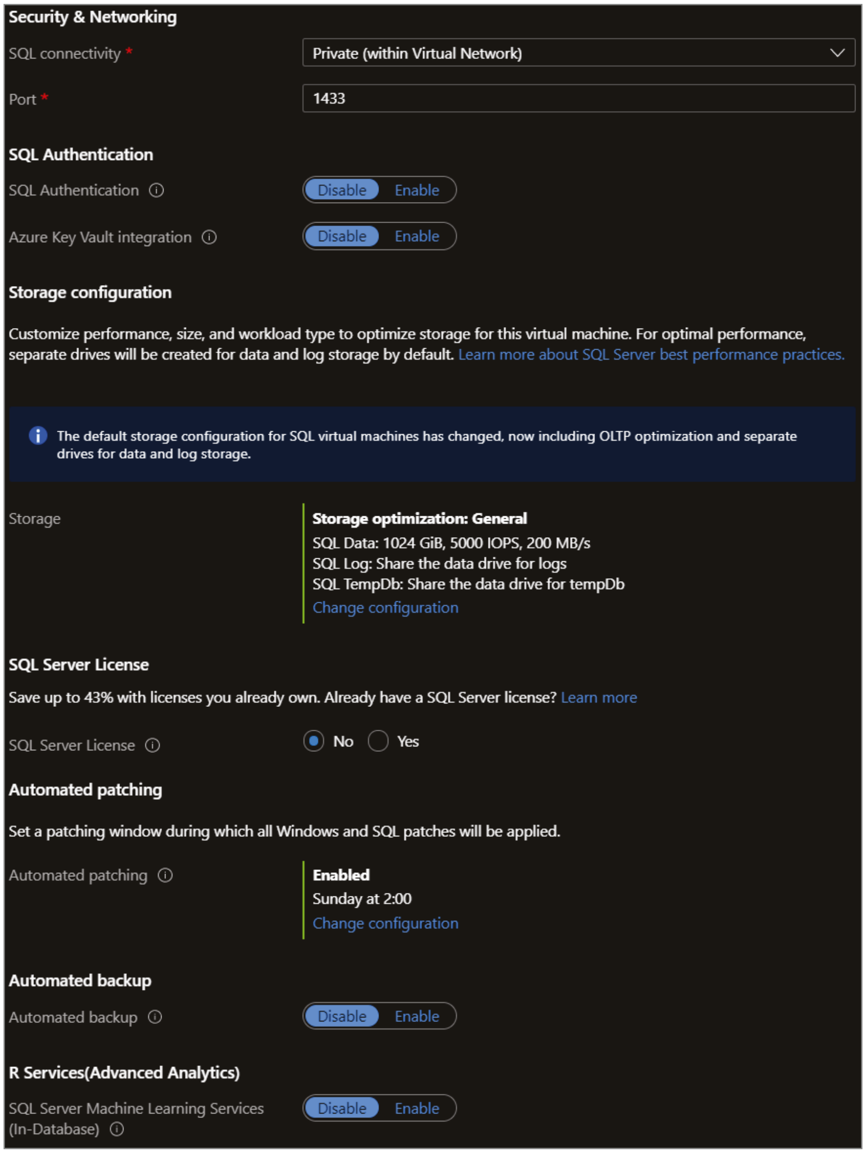
9. The SQL Server settings tab provide the following configuration options for the SQL Server instance hosted on the VM. Figure 2.8 illustrates a completed view of this tab.
   a. Choose the level of network isolation for SQL. The default for this option is limiting communication to applications that are connected to the VNet the VM is in. However, there are options to further lock the SQL Server instance down so that only applications in the VM can communicate to it and to relax security by allowing any application communicating over the public Internet to access it.
   b. Choose whether you would like to enable SQL Authentication and Azure Key Vault Integration.
   c. Review the default storage configuration for SQL Server's data, log, and tempdb files. Edit the configuration if the default options do not meet your requirements.
   d. Choose to apply an existing SQL Server license to reduce the cost of SQL.
   e. Choose a time window for when patches can be applied to the OS and SQL.
   f. Choose to enable automated backups if you would like to offload backup management to Azure.
   g. The last optional setting is for R Services. This will enable users to perform machine learning activities in SQL Server using the R language.
10. The Tags tab allows you to place tags on the resources deployed with the SQL Server VM. Tags are used to categorize resources for cost management purposes.
11. Finally, the Review + Create tab allows you to review the configuration choices made during the design process. If you are satisfied with the choices made for the VM, click the *Create* button to begin provisioning the SQL Server VM.

## Azure SQL Managed Instance

Migrating SQL Server workloads to Azure can provide more benefits than simply offloading hardware management. Organizations can also take advantage of PaaS benefits that remove the overhead of managing a virtual machine, such as the OS and the SQL Server instance from users. However, applications that require *instance-scoped* features will still need to be able to interact with the SQL Server instance. This leaves database architects with two options: (1) rearchitect the solution to use cloud native technologies in place of instance-scoped features, or( 2) migrate to a technology that supports these features. Prior to a few years back, this meant that organizations wishing to move to Azure needed to commit a lot of time to rebuilding the solution or move to SQL Server on a VM and manage the virtual machine and SQL Server–level maintenance such as upgrades. It is for these reasons that Microsoft introduced Azure SQL Managed Instance.

*Azure SQL Managed Instance*, or Azure SQL MI for short, is a PaaS database offering on Azure. It abstracts the OS but includes a SQL Server instance so that users can continue using their existing SQL Server processes without having to manage hardware or virtual machines. This makes it the ideal solution for customers looking to migrate many databases to Azure with as little effort as possible. Azure SQL MI also includes many system databases such as model, msdb, and tempdb. It can be used to host a distribution database for transactional replication, SSRS databases, and SSIS data catalog databases.

**FIGURE 2.8** Create a SQL virtual machine: Settings tab.



**Security & Networking**

SQL connectivity * Private (within Virtual Network)

Port * 1433

**SQL Authentication**

SQL Authentication: [✓ Disable] Enable

Azure Key Vault integration: [✓ Disable] Enable

**Storage configuration**

Customize performance, size, and workload type to optimize storage for this virtual machine. For optimal performance, separate drives will be created for data and log storage by default. Learn more about SQL Server best performance practices.

The default storage configuration for SQL virtual machines has changed, now including OLTP optimization and separate drives for data and log storage.

Storage

**Storage optimization: General**
SQL Data: 1024 GiB, 5000 IOPS, 200 MB/s
SQL Log: Share the data drive for logs
SQL TempDb: Share the data drive for tempDb
Change configuration

**SQL Server License**

Save up to 43% with licenses you already own. Already have a SQL Server license? Learn more

SQL Server License: [✓ No] Yes

**Automated patching**

Set a patching window during which all Windows and SQL patches will be applied.

Automated patching

**Enabled**
Sunday at 2:00
Change configuration

**Automated backup**

Automated backup: [✓ Disable] Enable

**R Services(Advanced Analytics)**

SQL Server Machine Learning Services (In-Database): [✓ Disable] Enable

NOTE While Azure SQL MI can host SSISDB and SSRS catalog databases, it cannot host SSIS packages or SSRS. If you wish to host these services on Azure, they will need to be hosted on a virtual machine. Another alternative is to use a more modern approach by migrating SSIS packages to Azure Data Factory and hosting SSRS paginated reports in Power BI.

The Azure SQL MI database engine uses the latest version of SQL Server Enterprise Edition, with updates and patches applied by Microsoft as they are made available. Azure SQL MI is nearly 100 percent compatible with on-premises SQL Server and offers support for instance-scoped features such as the SQL Server Agent, common language runtime (CLR), linked servers, Database Mail, and distributed transactions. It also includes a native VNet implementation to provide network isolation for the databases it hosts.

### Service Tiers

There are two service tiers available for Azure SQL MI:

- *General Purpose* is designed for applications with typical performance requirements.
- *Business Critical* is designed for applications with low latency and strict HA requirements. This tier uses a SQL Server Always On availability group for HA and enables one of the secondary nodes to be used for read-only workloads.

Table 2.3 outlines some of the key differences between the two tiers. The descriptions listed are for the Gen5 hardware version of Azure SQL MI.

**TABLE 2.3** Azure SQL MI service tier characteristics

| Feature | General Purpose | Business Critical |
|---|---|---|
| Number of vCores | 4, 8, 16, 24, 32, 40, 64, 80 | 4, 8, 16, 24, 32, 40, 64, 80 |
| Max Memory | 20.4 GB—408 GB (5.1 GB/vCore) | 20.4 GB—408 GB (5.1 GB/vCore) |
| Storage Type | High Performance Azure Blob storage | Local SSD storage |
| Max Instance Storage | 2 TB for 4 vCores<br>8 TB for other sizes | 1 TB for 4, 8, 16 vCores<br>2 TB for 24 vCores<br>4 TB for 32, 40, 64, 80 vCores |
| Max Number of Databases per Instance | 100 user databases | 100 user databases |
| Data/Log IOPS | Up to 30–40K IOPS per instance | 16K–320K (4000 IOPS/vCore) |
| Storage I/O Latency | 5–10 ms | 1–2 ms |

More information on the different Azure SQL MI service categories can be found at `https://docs.microsoft.com/en-us/azure/azure-sql/managed-instance/resource-limits#service-tier-characteristics`. Each of these service tiers falls under the vCore-based purchasing model and can be scaled up or down in the Azure Portal or through an automation script as workload requirements change.

The cost for Azure SQL MI can be reduced using a couple of different methods. First, organizations with existing SQL Server licenses can apply them to Azure SQL MI to reduce its cost. If an organization does not have or decides not to use existing licenses, they can choose to purchase reserved capacity. Like Azure Reserved Virtual Machine Instances for SQL Server on Azure VMs, reserved capacity allows organizations to commit to Azure SQL MI for one or three years. To purchase reserved capacity, you will need to specify the Azure region the Azure SQL MI will be deployed to, the service tier, and the length of the commitment.

### Network Isolation

An Azure SQL MI is required to be placed inside a VNet upon creation. On top of this requirement, the subnet that the Azure SQL MI is deployed to must be dedicated to hosting one or more Azure SQL MIs. This requirement restricts access to databases hosted on the Azure SQL MI to only applications that can communicate with that VNet. On-premises networks that host applications connecting to Azure SQL MI can use a VPN or Azure ExpressRoute to communicate with the VNet in Azure.

Deploying an Azure SQL MI to a subnet for the first time creates more than just the database engine. Along with the database engine, the deployment will create the following:

- A virtual cluster to host each Azure SQL MI that is deployed to that subnet. An Azure SQL MI is made up of a set of service components that are hosted on a dedicated set of virtual machines that are abstracted from the user and run inside the subnet. Together, these virtual machines form a virtual cluster.
- A network security group (NSG) to control access to the SQL Managed Instance data endpoint by filtering traffic on port 1433 and ports 11000–11999 when SQL Managed Instance is configured for redirect connections. The NSG will be associated with the subnet once it is provisioned.
- A User Defined Route (UDR) table to route traffic that has on-premises private IP ranges as a destination through the virtual network gateway or virtual network appliance (NVA). The UDR table will be associated with the subnet once it is provisioned.

The subnet will also be delegated to the Microsoft.Sql/managedInstances resource provider. See the section "Azure Resource Manager Templates" later in this chapter for more information on resource providers.

NOTE

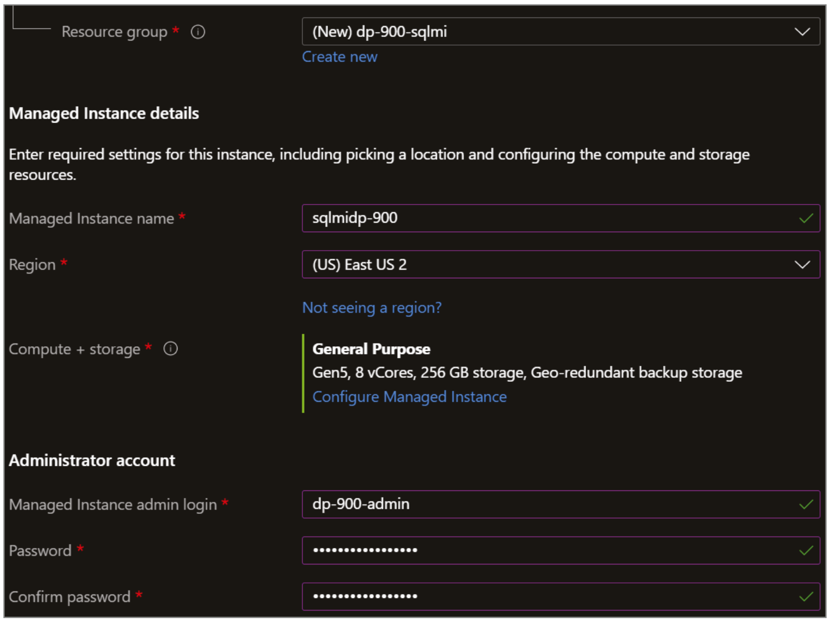
While knowing specific network requirement details for Azure SQL MI is not required for the DP-900 exam, it will be necessary to work with them. You can learn more about Azure SQL MI's network requirements at `https://docs.microsoft.com/en-us/azure/azure-sql/managed-instance/connectivity-architecture-overview`.

## Deploying through the Azure Portal

Use the following steps to create an Azure SQL MI through the Azure Portal:

1. Log into `portal.azure.com` and search for *SQL managed instances* in the search bar at the top of the page. Click *SQL managed instances* to go to the SQL managed instances page in the Azure Portal.
2. Click *Create* to start choosing the configuration options for your Azure SQL MI.
3. The *Create Azure SQL Database Managed Instance* page includes five tabs with different configuration options to tailor the Azure SQL MI to fit your needs. Let's start by exploring the options available in the Basics tab. Along with the following list that describes each option, you can view a completed example of this tab in Figure 2.9.
   a. Choose the subscription and resource group that will contain the Azure SQL MI and the databases deployed to the instance. You can create a new resource group on this page if you have not already created one.
   b. Enter a name for the Azure SQL MI.
   c. Choose the Azure region you wish to deploy the instance to.
   d. Choose a tier for the instance (i.e., General Purpose or Business Critical), the number of vCores, the storage amount, and the type of redundancy for backup storage.
   e. Set a username and password for the administrator account.

**FIGURE 2.9** Create Azure SQL Database Managed Instance: Basics tab.



Resource group * (New) dp-900-sqlmi
Create new

**Managed Instance details**

Enter required settings for this instance, including picking a location and configuring the compute and storage resources.

Managed Instance name * sqlmidp-900

Region * (US) East US 2
Not seeing a region?

Compute + storage * **General Purpose**
Gen5, 8 vCores, 256 GB storage, Geo-redundant backup storage
Configure Managed Instance

**Administrator account**

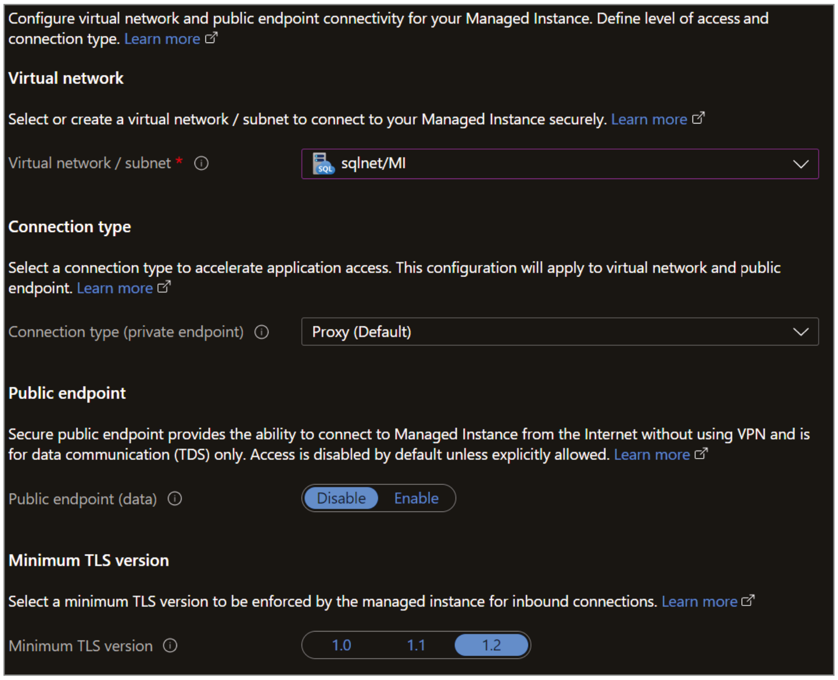
Managed Instance admin login * dp-900-admin

Password * ••••••••••••••••

Confirm password * ••••••••••••••••

4. The Networking tab provides the following network configuration options for the Azure SQL MI. A completed example of this tab can be seen in Figure 2.10.
   a. Choose the VNet and dedicated subnet that will host the Azure SQL MI.
   b. The next important component will be deciding whether you want to enable a public endpoint for the Azure SQL MI. Public endpoints are disabled by default to limit connectivity to applications that can communicate with the VNet that the Azure SQL MI is in.
   c. Choose the minimum TLS version that will be used to encrypt data in-transit for inbound connections. The default TLS version is 1.2 and should be left as is unless there are specific requirements for a lower version.

**FIGURE 2.10** Create Azure SQL Database Managed Instance: Networking tab.



Configure virtual network and public endpoint connectivity for your Managed Instance. Define level of access and connection type. Learn more

**Virtual network**

Select or create a virtual network / subnet to connect to your Managed Instance securely. Learn more

Virtual network / subnet * : sqlnet/MI

**Connection type**

Select a connection type to accelerate application access. This configuration will apply to virtual network and public endpoint. Learn more

Connection type (private endpoint): Proxy (Default)

**Public endpoint**

Secure public endpoint provides the ability to connect to Managed Instance from the Internet without using VPN and is for data communication (TDS) only. Access is disabled by default unless explicitly allowed. Learn more

Public endpoint (data): [✓ Disable] Enable

**Minimum TLS version**

Select a minimum TLS version to be enforced by the managed instance for inbound connections. Learn more

Minimum TLS version: 1.0 1.1 [✓ 1.2]

5. The Additional Settings tab provides options to change the collation, time zone, and maintenance window for the Azure SQL MI's underlying SQL Server database engine. It also includes an option to enable the instance as a secondary in an Azure SQL failover group for DR purposes.
6. The Tags tab allows you to place a tag on the Azure SQL MI for cost management.
7. Finally, the Review + Create tab allows you to review the configuration choices made during the design process. If you are satisfied with the choices made for the instance, click the *Create* button to begin provisioning the Azure SQL MI.

## Azure SQL Database

Modern applications that are built from the ground up with cloud native best practices rely on database platforms that are flexible and minimize the amount of administrative effort needed to manage the database. Administrators must be able to easily scale performance resources up or down to meet dynamic demand requirements at the most cost-optimal price point. Modern applications are typically designed not to need instance-scoped features that are available in a platform like SQL Server as these features can be implemented using other cloud native offerings. For example, Azure Data Factory, Azure Logic Apps, or Azure Automation can be used to automate when stored procedures or other tasks in the database are run, eliminating the need for SQL Server Agent jobs to perform custom maintenance tasks that are not natively handled by Microsoft.

*Azure SQL Database* is a fully managed PaaS database engine that is designed to serve cloud native applications. It abstracts both the OS and the SQL Server instance so that users can fully focus on application development. Management operations such as upgrades, patches, backups, HA, and monitoring are also handled behind the scenes without requiring any effort from the user. Azure SQL Database comes with a 99.99 percent availability guarantee, regardless of the deployment option or service tier. Just like Azure SQL MI, Azure SQL Database uses the latest version of SQL Server Enterprise Edition. In fact, the newest features of SQL Server are first released to Azure SQL Database before they are released to SQL Server.

Even though Azure SQL Database abstracts the physical SQL Server instance from the user, it still exposes a logical server. Unlike a physical server, the logical server does not expose any instance-scoped features. It instead serves as a parent resource for one or more Azure SQL databases, and maintains firewall, auditing, and threat detection rules for the databases it is associated with. The logical server also provides a connection endpoint for each Azure SQL Database associated with it for applications to use to connect to them.

Azure SQL Database provides two deployment options that allow organizations to optimize database performance and cost:

- A *single database* is a fully managed, isolated database. This option leverages all the resources (e.g., CPU and memory) allocated to it and is used when a modern application needs a single reliable database.
- An *elastic pool* is a collection of single databases with a shared set of resources, such as CPU or memory. Elastic pools are useful in scenarios where some databases are used more than others during different time periods. This will reduce the cost of these databases since they will be sharing the same pool of resources.

These options can be broken down further by the following purchasing models that are available for Azure SQL Database:

- The *DTU-based* purchasing model offers a fixed blend of CPU, memory, and IOPS. Each blended compute package is known as database transaction units (DTUs). The DTU-based purchasing model comes with a fixed amount of storage that varies for each service tier.

- The *vCore-based* purchasing model lets organizations choose how many virtual cores (vCores) they would like allocated. Service tiers using the vCore-based purchasing model allocate a fixed amount of memory per vCore that varies based on the hardware generation and compute option used. This purchasing model allows organizations to apply their existing SQL Server licenses to reduce the overall cost of the database. Reserved capacity is also exclusively available for the vCore-based purchasing model, allowing organizations to commit to Azure SQL Database for one or three years at a discounted rate. The vCore-based purchasing model provides two options for compute:
  - Provisioned compute allows organizations to deploy a specific service tier with a set amount of compute resources. Provisioned compute can be dynamically scaled manually or through an automation script.
  - Serverless compute allows organizations to specify a minimum and maximum vCore limit for a database. Databases configured to use serverless compute will automatically scale based on workload demand. It will also automatically pause databases during inactive periods and restart them when activity resumes to cut back on compute costs. This option is only available for single databases.

Deciding on which purchasing model to choose comes down to how much control over compute resources you would like to have. The DTU-based purchasing model offers a fixed combination of resources that allow organizations to start developing very quickly. The vCore-based purchasing model allows organizations to choose the amount of compute resources, or a range of compute resources in the case of serverless. This model also includes a more extensive selection of storage sizes as well as more cost-saving options with reserved capacity or existing licenses.

### Service Tiers

Azure SQL Database service tiers are different for each purchasing model. The DTU-based purchasing model offers Basic, Standard, and Premium tiers. Table 2.4 lists some of the common characteristics of these tiers.

**TABLE 2.4** DTU-based purchasing model service tier characteristics

| Characteristic | Basic | Standard | Premium |
|---|---|---|---|
| DTUs | 5 | S0: 10<br>S1: 20<br>S2: 50<br>S3: 100<br>S4: 200<br>S6: 400<br>S7: 800<br>S9: 1600<br>S12: 3000 | P1: 125<br>P2: 250<br>P4: 500<br>P6: 1000<br>P11: 1750<br>P15: 4000 |

**TABLE 2.4** DTU-based purchasing model service tier characteristics *(continued)*

| Characteristic | Basic | Standard | Premium |
|---|---|---|---|
| Included Storage | 2 GB | 250 GB | P1–P6: 500 GB<br>P11 and above: 4 TB |
| Maximum Storage | 2 GB | S0–S2: 250 GB<br>S3 and above: 1 TB | P1–P6: 1 TB<br>P11 and above: 4 TB |
| Maximum backup retention | 7 days | 35 days | 35 days |
| CPU | Low | Low, Medium, High | Medium, High |
| IOPS | 1–4 IOPS per DTU | 1–4 IOPS per DTU | >25 IOPS per DTU |
| IO Latency | 5 ms (read), 10 ms (write) | 5 ms (read), 10 ms (write) | 2 ms (read/write) |
| Columnstore Indexes | N/A | S3 and above | Supported |
| In-Memory OLTP | N/A | N/A | Supported |

The vCore-based purchasing model offers the following three service tiers:

- *General Purpose* is used for most business workloads. This tier offers balanced compute and storage options.
- *Business Critical* is used for business applications that require high I/O performance. It is also the best option for applications that require high resiliency to outages by leveraging a SQL Server Always On availability group for HA.
- *Hyperscale* is used for very large OLTP databases (>4 TB) and can automatically scale storage and compute. Hyperscale databases use local SSDs for local buffer-pool cache and data storage. Long-term data storage is done with remote storage.

Table 2.5 lists the common characteristics for the vCore-based purchasing model service tiers:

**TABLE 2.5** vCore-based purchasing model service tier characteristics

| Characteristic | General Purpose | Business Critical | Hyperscale |
|---|---|---|---|
| Storage | Uses remote storage. Provisioned Compute: 5 GB–4 TB Serverless Compute: 5 GB–4 TB | Uses local SSD storage Provisioned Compute: 5 GB–4 TB | Supports up to 100 TB |
| Availability | 1 replica, no read-scale replicas | 3 replicas, 1 read-scale replica | 1 read-write replica, 0–4 read-scale replicas |
| In-Memory | Not Supported | Supported | Partial Support |

NOTE Resource limits for the vCore-based purchasing model such as the number of vCores, amount of memory, IO latency, and maximum IOPS depend on the type of hardware chosen. See `https://docs.microsoft.com/en-us/azure/azure-sql/database/resource-limits-vcore-single-databases` for the resource limits related to hardware available in the vCore-based purchasing model.

### Network Isolation

Unlike SQL Server on a VM and Azure SQL MI, a logical server for an Azure SQL Database does not come with a built-in private endpoint. This means that an Azure SQL Database is not isolated within a VNet by default. Network isolation for Azure SQL Database can instead be achieved by limiting access to the logical server's public endpoint through the server's firewall, restricting access to only services in a specific VNet or subnet, or explicitly adding a private endpoint that is associated with a subnet in a VNet.

Public endpoint access can be limited using the following settings:

- *Allow Azure Services* allows all resources hosted on Azure, such as an Azure VM or Azure Data Factory, to communicate with databases associated with the logical server. This setting is turned off by default, as turning it on typically provides database access to more resources than what is needed.
- *IP firewall rules* open port 1433 (the default port SQL Server listens on) to a specific IP address or a range of IP addresses. Firewall rules can be set at the server level to allow access to all databases associated with a logical server or at the database level to only allow access to a specific database.

NOTE Server-level IP firewall rules can be created using the Azure Portal, Azure PowerShell, the Azure CLI, the Azure REST API, and T-SQL. Database-level firewall rules can only be created and managed with T-SQL.

Private access to the logical server can also be enabled so that database connectivity is restricted to specific VNets. This type of access can be enabled using one of the following settings:

- *Virtual network firewall rules* restrict access to databases associated with a logical server to traffic using the private IP range of a VNet. Application traffic coming from a specific subnet in a VNet can be switched from using public IP addresses to private IP addresses by adding the Microsoft.Sql service endpoint to the subnet. The subnet can then be added as a virtual network rule in the logical server to allow traffic from that subnet to connect to databases associated with the logical server.
- *Private Link* is a service in Azure that allows you to add a private endpoint to a logical server. Private endpoints are private IP addresses within a specific subnet in a VNet. Once a private endpoint is attached, connectivity will be limited to other applications in the VNet or applications that can connect to the VNet through VNet peering, VPN, or Azure ExpressRoute.

NOTE

Relational databases covered later in this chapter use the same methods for network isolation as Azure SQL Database. IP or virtual network firewall rules can be set at the logical server level or private endpoints can be attached to the logical server to provide network isolation for all databases associated to that logical server. However, unlike with Azure SQL Database, these rules can only be applied at the server level and not at the individual database level. These services include Azure Synapse Analytics dedicated SQL pools, Azure Database for MySQL, Azure Database for PostgreSQL, and Azure Database for MariaDB.

### Deploying Through the Azure Portal

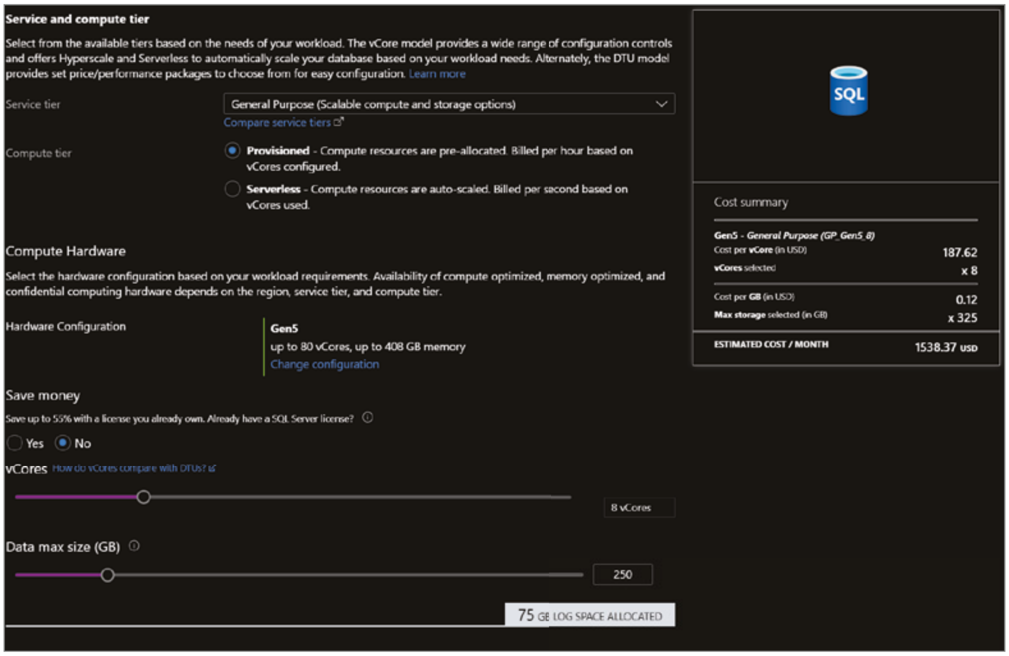
Use the following steps to create an Azure SQL Database through the Azure Portal:

1. Log into `portal.azure.com` and search for *SQL databases* in the search bar at the top of the page. Click *SQL databases* to go to the SQL databases page in the Azure Portal.
2. Click *Create* to start choosing the configuration options for your Azure SQL Database.
3. The *Create SQL Database* page includes six tabs with different configuration options to tailor the Azure SQL Database to fit your needs. Let's start by exploring the options available in the Basics tab. Along with the following list that describes each option, you can view a completed example of this tab in Figure 2.12.
   a. Choose the subscription and resource group that will contain the Azure SQL Database. You can create a new resource group on this page if you have not already created one.
   b. Enter a name for the Azure SQL Database.
   c. Choose the logical server you wish to deploy the database to. You can create a new logical server on this page if there is not one already available. The logical server chosen will dictate which region the database will be deployed to. Note that creating a new logical server will also require you to set a username and password for the administrator account.

**d.** Choose whether the database will be a part of an elastic pool.

**e.** Click *Configure database* to choose the purchasing model and service tier. If you choose one of the vCore-based purchasing model service tiers, you will be given the option to apply existing SQL Server licenses, choose the number of vCores, and set the maximum amount of storage allocated for data. Choosing a DTU-based purchasing model service tier will give you options to change the number of DTUs allocated to the database and the maximum amount of storage allocated for data. Figure 2.11 is an example of completed configuration for a General Purpose database. As you can see, the database configuration comes with a monthly cost estimate.

**FIGURE 2.11** Configuring an Azure SQL Database



**Service and compute tier**

Select from the available tiers based on the needs of your workload. The vCore model provides a wide range of configuration controls and offers Hyperscale and Serverless to automatically scale your database based on your workload needs. Alternately, the DTU model provides set price/performance packages to choose from for easy configuration. Learn more

Service tier: General Purpose (Scalable compute and storage options)

Compare service tiers

Compute tier: [✓ **Provisioned** - Compute resources are pre-allocated. Billed per hour based on vCores configured.] **Serverless** - Compute resources are auto-scaled. Billed per second based on vCores used.

Compute Hardware

Select the hardware configuration based on your workload requirements. Availability of compute optimized, memory optimized, and confidential computing hardware depends on the region, service tier, and compute tier.

Hardware Configuration: **Gen5**
up to 80 vCores, up to 408 GB memory
Change configuration

Save money

Save up to 55% with a license you already own. Already have a SQL Server license?

Yes [✓ No]

vCores How do vCores compare with DTUs?

8 vCores

Data max size (GB)

250

75 GB LOG SPACE ALLOCATED

Cost summary

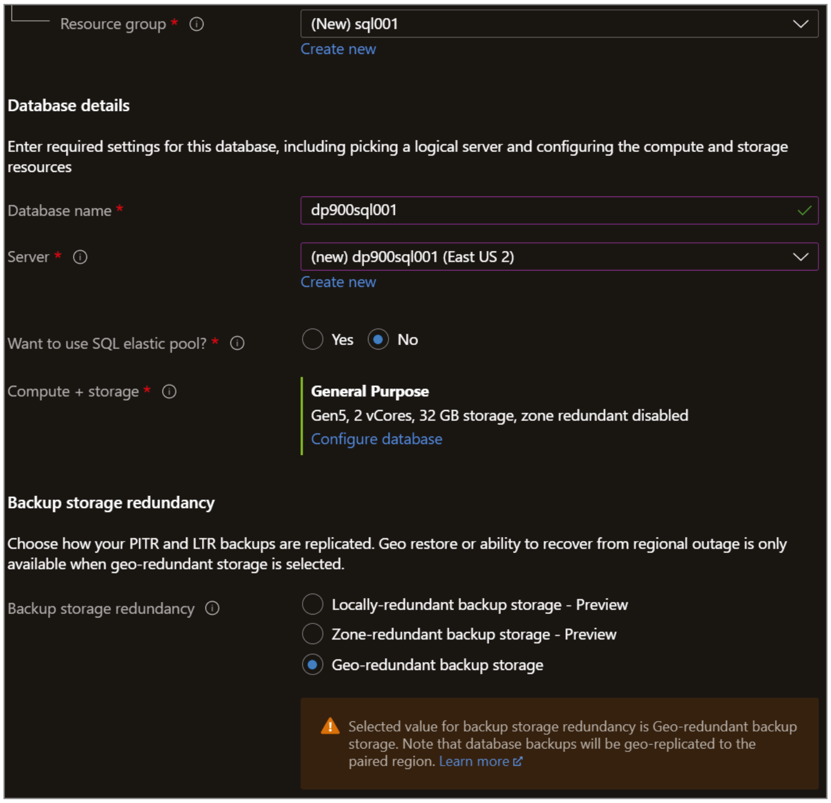
**Gen5** - ***General Purpose (GP_Gen5_8)***

| | |
|---|---|
| Cost per **vCore** (in USD) | 187.62 |
| **vCores** selected | x 8 |
| Cost per **GB** (in USD) | 0.12 |
| **Max storage** selected (in GB) | x 325 |
| **ESTIMATED COST / MONTH** | 1538.37 USD |

**f.** Choose the redundancy tier for database backups.

**4.** The Networking tab allows you to configure network access and connectivity for your logical server if you are creating a new one. If you are deploying the database to an existing logical server, then most of the options will be grayed out as it will be taking on the existing state of the server. A completed example of configuring a new logical server can be seen in Figure 2.13.

**FIGURE 2.12** Create Azure SQL Database: Basics tab.



Resource group * (New) sql001
Create new

**Database details**

Enter required settings for this database, including picking a logical server and configuring the compute and storage resources

Database name * dp900sql001

Server * (new) dp900sql001 (East US 2)
Create new

Want to use SQL elastic pool? * Yes [✓ No]

Compute + storage * **General Purpose**
Gen5, 2 vCores, 32 GB storage, zone redundant disabled
Configure database

**Backup storage redundancy**

Choose how your PITR and LTR backups are replicated. Geo restore or ability to recover from regional outage is only available when geo-redundant storage is selected.

Backup storage redundancy Locally-redundant backup storage - Preview / Zone-redundant backup storage - Preview / [✓ Geo-redundant backup storage]

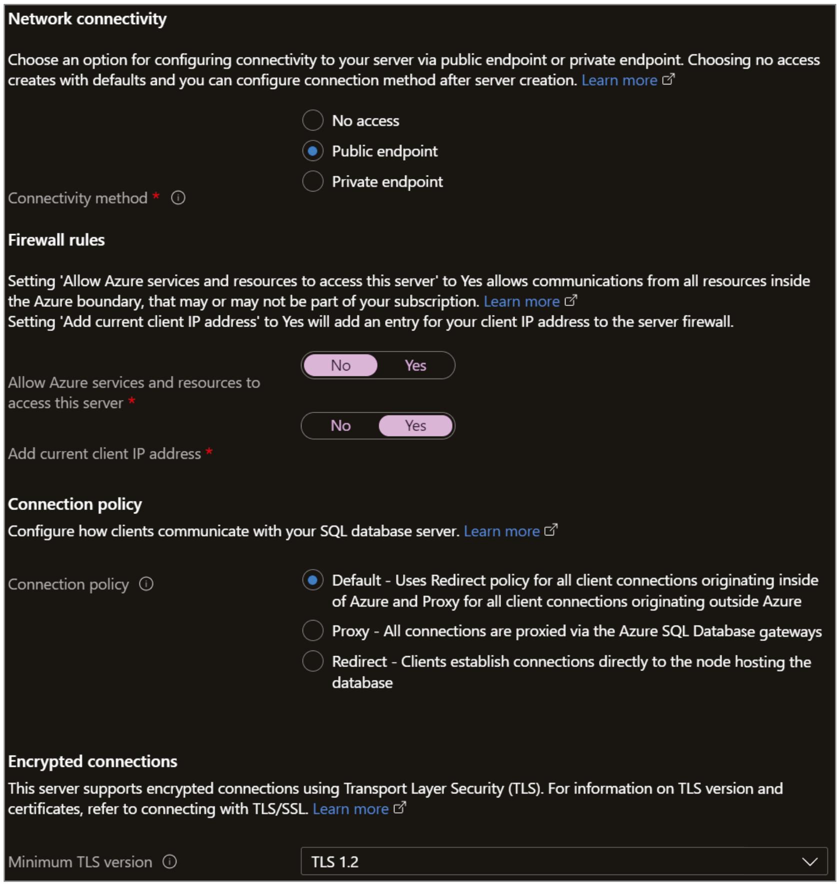
Selected value for backup storage redundancy is Geo-redundant backup storage. Note that database backups will be geo-replicated to the paired region. Learn more

a. There are three options available for network connectivity. The first option, *No Access*, allows you to continue configuring your database without needing to configure any connectivity until after it is provisioned. *Public Endpoint* will display a new set of options specific to the logical server's firewall. These will allow you to allow or deny Azure services and the client IP address you are deploying the logical server from access to the databases on the server. The final option, *Private Endpoint*, will allow you to associate a private IP address from a VNet to the logical server. This will isolate the databases within a VNet, allowing connectivity only to applications that can communicate with the VNet.

b. Choose how client applications will communicate with the logical server.

c. Choose the minimum TLS version that will be used to encrypt data in-transit for inbound connections. The default TLS version is 1.2 and should be left as is unless there are specific requirements for a lower version.

**FIGURE 2.13** Create Azure SQL Database: Networking tab.



**Network connectivity**

Choose an option for configuring connectivity to your server via public endpoint or private endpoint. Choosing no access creates with defaults and you can configure connection method after server creation. Learn more

Connectivity method *: No access [✓ Public endpoint] Private endpoint

**Firewall rules**

Setting 'Allow Azure services and resources to access this server' to Yes allows communications from all resources inside the Azure boundary, that may or may not be part of your subscription. Learn more
Setting 'Add current client IP address' to Yes will add an entry for your client IP address to the server firewall.

Allow Azure services and resources to access this server *: [✓ No] Yes

Add current client IP address *: No [✓ Yes]

**Connection policy**

Configure how clients communicate with your SQL database server. Learn more

Connection policy: [✓ Default - Uses Redirect policy for all client connections originating inside of Azure and Proxy for all client connections originating outside Azure] Proxy - All connections are proxied via the Azure SQL Database gateways; Redirect - Clients establish connections directly to the node hosting the database

**Encrypted connections**

This server supports encrypted connections using Transport Layer Security (TLS). For information on TLS version and certificates, refer to connecting with TLS/SSL. Learn more

Minimum TLS version: TLS 1.2

5. The Security tab allows you to choose if you would like to use Azure Defender for SQL to provide advanced threat protection for your data.
6. The Additional Settings tab allows you to start your database as a blank database, from a backup, or from a sample provided by Microsoft. You can also choose if you would like to change the default collation for the database and the default maintenance window.
7. The Tags tab allows you to place a tag on the Azure SQL Database for cost management.
8. Finally, the Review + Create tab allows you to review the configuration choices made during the design process. If you are satisfied with the choices made for the instance, click the *Create* button to begin provisioning the Azure SQL Database.

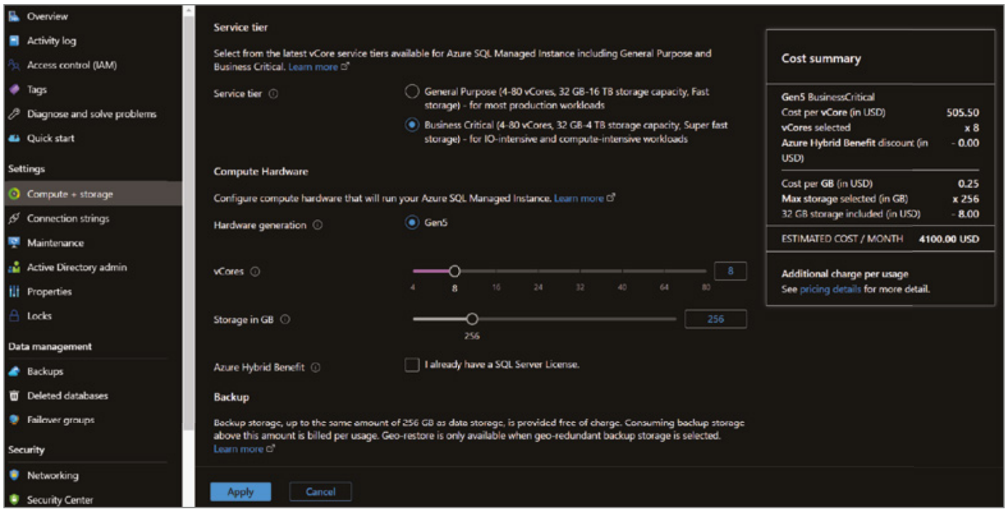
TIP Keep a note of the name of the logical server. This will be important later in this chapter when we walk through adding an Azure Active Directory user or group as an administrator account.

## Scaling PaaS Azure SQL in the Azure Portal

Scaling Azure SQL MI or Azure SQL Database resources up or down depending on workload demand, also known as vertical scale, is very easy in the Azure Portal. The need to vertically scale can result from performance degradation due to a lack of compute resources or overallocated compute resources that result in unnecessary expenses. The speed at which users can vertically scale compute and storage resources through the Azure Portal allows organizations to react very quickly to a change in workload demand. Since this process is the same for Azure SQL MI and Azure SQL Database, this section will detail how to scale an Azure SQL MI as an example. The only difference between the two is that you will need to go to the SQL databases page to scale your Azure SQL Database instead of the SQL managed instances page.

To scale an Azure SQL MI, go to the SQL managed instances page in the Azure Portal. Click your recently created Azure SQL MI and click the *Compute + storage* option under *Settings*. This page will allow you to change the service tier, number of vCores, and amount of storage allocated to the instance. The page will also update the cost summary for the instance as you change different configuration settings. Figure 2.14 illustrates an example of this process.

**FIGURE 2.14** Scaling an Azure SQL MI



Service tier

Select from the latest vCore service tiers available for Azure SQL Managed Instance including General Purpose and Business Critical. Learn more

Service tier: General Purpose (4-80 vCores, 32 GB-16 TB storage capacity, Fast storage) - for most production workloads [✓ Business Critical (4-80 vCores, 32 GB-4 TB storage capacity, Super fast storage) - for IO-intensive and compute-intensive workloads]

Compute Hardware

Configure compute hardware that will run your Azure SQL Managed Instance. Learn more

Hardware generation: [✓ Gen5]

vCores: 8 (4, 8, 16, 24, 32, 40, 64, 80)

Storage in GB: 256

Azure Hybrid Benefit: ☐ I already have a SQL Server License.

Backup

Backup storage, up to the same amount of 256 GB as data storage, is provided free of charge. Consuming backup storage above this amount is billed per usage. Geo-restore is only available when geo-redundant backup storage is selected. Learn more

Cost summary

| | |
|---|---|
| Gen5 BusinessCritical | |
| Cost per vCore (in USD) | 505.50 |
| vCores selected | x 8 |
| Azure Hybrid Benefit discount (in USD) | - 0.00 |
| Cost per GB (in USD) | 0.25 |
| Max storage selected (in GB) | x 256 |
| 32 GB storage included (in USD) | - 8.00 |
| ESTIMATED COST / MONTH | 4100.00 USD |

Additional charge per usage
See pricing details for more detail.

## Business Continuity for PaaS Azure SQL

Azure manages backups for Azure SQL Database and Azure SQL MI databases by creating a full backup every week, differential backups every 12 to 24 hours, and transaction log backups every 5 to 10 minutes. These backups are stored in geo-redundant Azure Blob storage and are replicated to a separate Azure region. Backups are kept for 7 to 35 days, depending on the service tier and the retention settings set by an administrator. Long-term backup retention (LTR) can also be enabled to retain full database backups for up to 10 years.

Database backups can be restored to Azure SQL Database or Azure SQL MI by performing a *point-in-time restore (PITR)*. PITR can restore a backup from an existing database or a deleted database. Database backups taken from Azure SQL MI can be restored to the same Azure SQL MI with a different database name or a different Azure SQL MI. This can be done through the Azure Portal, the Azure command-line interface (CLI), or Azure PowerShell.

High availability for Azure SQL Database and Azure SQL MI differs depending on the service tier being used. The following sections outline the high availability architectures used by each service tier of Azure SQL Database and Azure SQL MI.

### Basic, Standard, and General Purpose

High availability for the Basic, Standard, and General Purpose tiers of Azure SQL Database and the General Purpose tier of Azure SQL MI is accomplished through the standard availability model. This includes the following two layers:

- A stateless compute layer that runs the `sqlservr.exe` process and contains only ephemeral data such as data stored in tempdb. This is operated by Azure Service Fabric, which will move `sqlservr.exe` to another stateless compute node in the event of a database or OS upgrade or a failure. This process guarantees 99.99 percent availability but could result in performance degradation since `sqlservr.exe` will start with a cold cache after a failover.
- A stateful data layer with the data files stored in Azure Blob storage which has built-in HA.

### Premium, Business Critical, and Hyperscale

High availability for the Premium and Business Critical tiers of Azure SQL Database and the Business Critical tier of Azure SQL MI is accomplished through the Premium availability model. This model uses a SQL Server Always On AG for HA and deploys an additional three or four nodes behind the scenes to act as secondaries in the AG. The AG synchronously replicates compute and storage from the primary node to each of the secondaries. This ensures that the secondaries are in sync with the primary node before fully committing each transaction. Azure Service Fabric will automatically initiate a failover to one of the secondaries if the primary node experiences any downtime. This will ensure that anyone using the database will not notice the failover. An added benefit of this configuration is that one of the secondaries can be used for read-only workloads. This increases performance by eliminating resource contention between read-only and write operations.

NOTE

Disaster recovery for Azure SQL database is achieved by zone redundancy. This process replicates the HA model used to three different availability zones in the same region. Disaster recovery for Azure SQL MI can be achieved by adding the Azure SQL MI to a failover group with another Azure SQL MI that is hosted in a different region. More information on failover groups can be found at `https://docs.microsoft.com/en-us/azure/azure-sql/database/auto-failover-group-overview?tabs=azure-powershell#best-practices-for-sql-managed-instance`.

## Azure Synapse Analytics Dedicated SQL Pools

Azure Synapse Analytics dedicated SQL pools is a PaaS relational database engine that is optimized for data warehouse workloads. Dedicated SQL pools use a *scale-out* MPP architecture to process very large amounts of data. This means that data is sharded into multiple distributions and processed across one or more compute nodes. To do this, dedicated SQL pools separate compute and storage by using a SQL engine to perform computations and Azure Storage to store the data. Even though data is stored in Azure Blob storage, dedicated SQL pools serve data to users in a relational format as tables or views.

Dedicated SQL pools shard data into 60 distributions across one or more compute nodes. There are three different distribution patterns to consider when creating tables or materialized views. The most optimal choice is going to depend on the size and nature of the table or materialized view. They include the following distribution patterns:

- Hash distribution uses a hash function to deterministically assign each row to a distribution. In the table or view definition, one of the columns is designated as the distribution column. The most optimal distribution columns have a high number of distinct values and an even amount of data skew. Hash distribution is the best option for large fact and dimension tables as it provides the best performance for joins and aggregations on large tables.
- Round-robin distribution is the simplest distribution pattern as it evenly shards data randomly across distributions. Data is loaded quickly to a table or view using round-robin distribution but it can cause performance issues as data is not organized in the most optimal manner across each distribution. Typical use cases for round-robin distribution include staging tables or using it if there are no columns with highly distinct values.
- Replicated tables or materialized views cache a full copy of the table or materialized view on the first distribution on each compute node. This provides the fastest query performance as data does not need to shuffle from one distribution to another when aggregated or joined. Because extra storage is required, replicated tables and materialized views are recommended for small tables or tables that contain static values.

Distribution design should be carefully considered since data distribution results in data being physically stored in different locations. For example, round-robin distribution tables or poorly chosen distribution columns on hash distributed tables could result in a lot of data *shuffling* when the data is queried. The more that data needs to be shuffled, the more time the query will take to complete.

Just as with Azure SQL Database, it is easy to scale a dedicated SQL pool up or down depending on workload demands through the Azure Portal, PowerShell, or T-SQL. Service level objectives (SLOs) represent the scalability setting of a dedicated SQL pool and determine the cost and performance level as well as the number of compute nodes allocated. These are measured by compute Data Warehouse Units (cDWUs) which are bundled compute units of CPU, memory, and I/O. Table 2.6 lists the available dedicated SQL pool SLOs.

**TABLE 2.6** Dedicated SQL pool service level objectives

| Performance Level | Compute Nodes | Distributions per Compute Node | Memory (GB) |
|---|---|---|---|
| DW100c | 1 | 60 | 60 |
| DW200c | 1 | 60 | 120 |
| DW300c | 1 | 60 | 180 |
| DW400c | 1 | 60 | 240 |
| DW500c | 1 | 60 | 300 |
| DW1000c | 2 | 30 | 600 |
| DW1500c | 3 | 20 | 900 |
| DW2000c | 4 | 15 | 1,200 |
| DW2500c | 5 | 12 | 1,500 |
| DW3000c | 6 | 10 | 1,800 |
| DW5000c | 10 | 6 | 3,000 |
| DW6000c | 12 | 5 | 3,600 |
| DW7500c | 15 | 4 | 4,500 |
| DW10000c | 20 | 3 | 6,000 |
| DW15000c | 30 | 2 | 9,000 |
| DW30000c | 60 | 1 | 18,000 |

NOTE Dedicated SQL pools are one part of the broader Azure Synapse Analytics suite of analytical components and will be discussed further in Chapter 5, "Modern Data Warehouses in Azure." This includes how to deploy a dedicated SQL pool through the Azure Portal in an Azure Synapse Analytics workspace.

## Open-Source Databases in Azure

While SQL Server is a very popular relational database offering, there are several organizations that rely on open-source database platforms to store their relational data. Open-source database platforms can be deployed quickly at very little cost, enabling organizations to stand up a storage platform for their applications with little overhead. However, on-premises open-source database deployments still require organizations to manage hardware, OS, and database engine maintenance. For this reason, Azure offers three PaaS options for hosting open-source databases. These include Azure Database for MySQL, Azure Database for MariaDB, and Azure Database for PostgreSQL. Just like Azure SQL Database, these offerings come with native high availability, automatic patching, automatic backups, and automatic threat protection.

Each of these offerings use the vCore-based purchasing model and includes the following three service tiers:

- *Basic*—Workloads that require light compute and I/O performance, such as development and test environments
- *General Purpose*—Most business workloads that require a balance of compute and memory, with scalable I/O performance
- *Memory Optimized*—Workloads that require high performance and in-memory capabilities

Azure Database for MySQL and Azure Database for PostgreSQL include two deployment options: Single Server and Flexible Server.

- Single Server is a fully managed database service that manages the database engine, handling database and OS patches, automatic backup schedules, and high availability. This option is best suited for modern applications that use cloud native design practices.
- Flexible Server gives users more granular control over the management of the database engine. It allows users to configure high availability within one availability zone or across multiple availability zones. Users can also stop and start the server and set a burstable compute tier for workloads that do not always need a fixed compute capacity.

The following sections will only cover Single Server as Flexible Server is still in preview and is not a focus of the DP-900 exam.

NOTE Single Server is the only deployment option available for Azure Database for MariaDB as of the writing of this book.

Discount pricing for each of these options is available by prepaying for compute resources. Reserved capacity allows users to purchase a one-year term for Azure Database for PostgreSQL and one- or three-year terms for Azure Database for MySQL and Azure Database for MariaDB. As with Azure SQL, the number of vCores will need to be known beforehand as these are the resources that are purchased.

## Azure Database for MySQL

MySQL is an open-source relational database engine that is very similar to SQL Server. Users can issue queries to a MySQL database using SQL, with some nuanced syntax differences versus how Microsoft SQL Server implements SQL.

Azure Database for MySQL is a PaaS relational database offering based on the MySQL Community Edition. Supported versions of the MySQL database engine include 5.6, 5.7, and 8.0. Azure Database for MySQL includes the resource configuration options for each pricing tier shown in Table 2.7.

**TABLE 2.7** Azure Database for MySQL service tier resource options

| Feature | Basic | General Purpose | Memory Optimized |
|---|---|---|---|
| Number of vCores | 1, 2 | 2, 4, 8, 16, 32, 64 | 2, 4, 16, 32 |
| Amount of Memory per vCore | 2 GB | 5 GB | 10 GB |
| Storage Size | 5 GB to 1 TB | 5 GB to 16 TB | 5 GB to 16 TB |

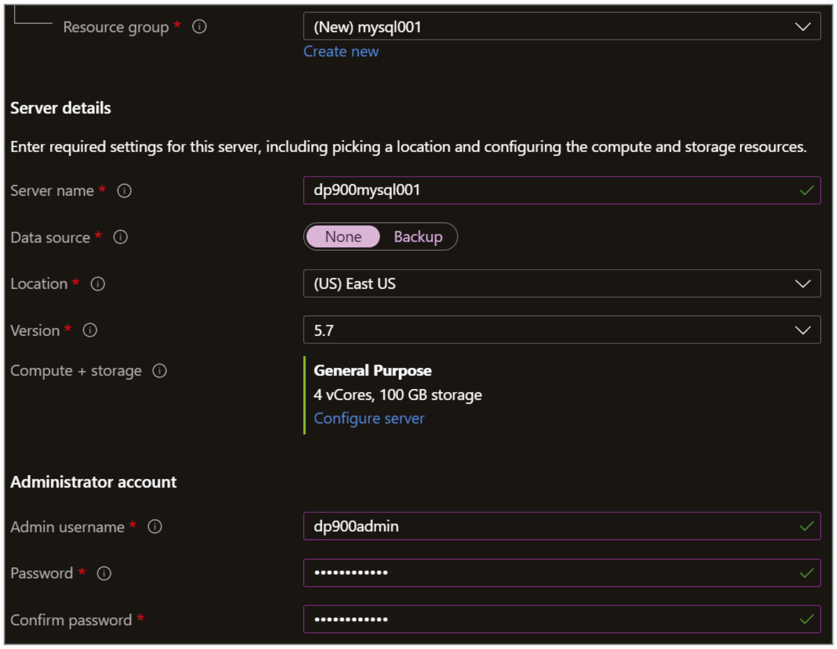
### Deploying Through the Azure Portal

Azure Database for MySQL through the Azure Portal is very similar to how you would deploy an Azure SQL Database.

1. Log into `portal.azure.com` and search for *Azure Database for MySQL servers* in the search bar at the top of the page. Click *Azure Database for MySQL servers* to go to the Azure Database for MySQL servers page in the Azure Portal.
2. Click *Create* to start choosing the configuration options for your Azure Database for MySQL server.
3. The next page will allow you to select which deployment option you would like to use. This example will demonstrate how to configure a Single Server deployment. Click *Create* under the *Single server* option to continue.
4. The *Create MySQL server* page includes six tabs with different configuration options to tailor the Azure Database for MySQL server to fit your needs. Let's start by exploring the options available in the Basics tab. Along with the following list that describes each option, you can view a completed example of this tab in Figure 2.15.

a. Choose the subscription and resource group that will contain the Azure Database for MySQL server. You can create a new resource group on this page if have not already created one.
b. Enter a server name.
c. Choose to start without any databases associated with the server or to restore a database backup to the server as it is being deployed.
d. Choose the Azure region that the server will be located in.
e. Choose the MySQL database engine version.
f. Choose the service tier, number of vCores, storage amount, backup retention period, and backup redundancy. Note that storage cannot be scaled down once the server is deployed.
g. Note that creating a new logical server will also require you to set a username and password for the administrator account.

**FIGURE 2.15** Create MySQL server: Basics tab.



Resource group * (New) mysql001
Create new

Server details

Enter required settings for this server, including picking a location and configuring the compute and storage resources.

Server name * dp900mysql001
Data source * [✓ None] Backup
Location * (US) East US
Version * 5.7
Compute + storage: General Purpose
4 vCores, 100 GB storage
Configure server

Administrator account

Admin username * dp900admin
Password * ••••••••••••
Confirm password * ••••••••••••

5. The Additional Settings tab allows you to enable double encryption if it is required. This setting will add an additional infrastructure encryption layer on top of the database and database backup encryption layer.

6. The Tags tab allows you to place a tag on the Azure Database for MySQL server for cost management.
7. Finally, the Review + Create tab allows you to review the configuration choices made during the design process. If you are satisfied with the choices made for the instance, click the *Create* button to begin provisioning the Azure Database for MySQL server.

### Azure Database for MariaDB

MariaDB is another open-source relational database platform that is a fork of MySQL. In fact, the founders of MariaDB were the original founders of MySQL. There are some performance enhancements made to the query optimizer and the storage engine, but most of the core functionality is the same as MySQL. More information on MariaDB can be found at `https://mariadb.org`.

Azure Database for MySQL is a PaaS relational database offering based on the MariaDB Community Edition. Supported versions of the MariaDB database engine include 10.2 and 10.3.

Azure Database for MariaDB includes the same service tier resource configurations as Azure Database for MySQL. It also includes most of the same configuration options as Azure Database for MySQL when deploying it through the Azure Portal. The only differences are that Azure Database for MariaDB does not require you to select Single Server or Flexible Server and it does not have an Additional Settings tab.

### Azure Database for PostgreSQL

PostgreSQL is an open-source object-relational database system that uses SQL for native queries. It uses a robust feature set with standard and complex data types, including these:

- *Primitives:* Integer, numeric, string, Boolean
- *Document:* JSON/JSONB, XML, key-value pair
- *Geometry:* Point, line, circle, polygon

The PostgreSQL database engine is also highly extensible, allowing users to define their own data types and custom functions with its proprietary language PL/PGSQL or other common development languages like Perl and Python. There are also custom extensions available that solve specific business problems, such as the PostGIS geospatial database extender. This extension adds geospatial-specific functionality that effectively turns PostgreSQL into a spatial database management system. More information about PostgreSQL and PostGIS can be found at `www.postgresql.org/about`.

Azure Database for PostgreSQL is a PaaS relational database offering based on the PostgreSQL Community Edition. Supported versions of the PostgreSQL database engine include 9.6, 10, and 11 for Single Server as well as 11, 12, and 13 for Flexible Server. Azure Database for PostgreSQL includes the same service tier resource configurations as Azure Database for MySQL for its Single Server and Flexible Server deployment models. It also includes the same configuration options as Azure Database for MySQL when deploying it through the Azure Portal.

Along with the Single Server and Flexible Server deployment models, Azure Database for PostgreSQL also includes a Hyperscale deployment option. Hyperscale (Citus) horizontally scales queries across multiple nodes through data sharding. This deployment option is typically used for multi-tenant applications that require greater scale and performance, such as real-time operational and high throughput transactional workloads. Azure Database for PostgreSQL Hyperscale (Citus) supports versions 11, 12, and 13 of the PostgreSQL database engine.

## Management Tasks for Relational Databases in Azure

While Azure removes many of the rigid maintenance demands that come with managing an on-premises relational database environment, there are still several management tasks that must be handled. Failing to give these tasks the proper attention will result in poor database performance or, even worse, potential security risks. These common management tasks are included:

- Managing the deployment of the database through the Azure Portal or with automation scripts
- Migrating existing on-premises relational data to the new environment in Azure
- Maintaining data security through network isolation, access management, threat protection, and data encryption

There are also times that connectivity issues arise and must be troubleshooted. These can be the result of unexpected and expected behavior depending on how the service is configured in Azure. The following sections detail these tasks as well as some of the tools that can be used for database management.

### Deployment Scripting and Automation

Cloud environments such as Azure greatly reduce the complexity involved in standing up a relational database. Tasks such as procuring hardware, installing network devices, and reserving capacity in a datacenter that previously required months of planning and implementation are reduced to a matter of minutes. Relational databases in Azure can also be scaled down or deleted just as quickly when they are not needed, allowing organizations to cut costs on services not being used.

In the previous sections we discussed how organizations can leverage the Azure Portal to manually deploy a relational database service. While this makes it easy to get started with a database in a single environment, it is not the most practical solution for deploying databases to multiple environments. Most organizations use several application development life cycle stages such as development, test, and quality assurance to make sure each release of an application meets a specific level of satisfaction before being pushed to production.

Cloud-based services make this process easy by allowing development teams to package their infrastructure requirements in automation scripts that describe each service to be deployed and their desired configuration. These scripts can be parameterized to meet the cost and performance needs of different environments used in an application's development life cycle.

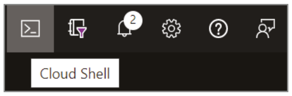
Azure offers three primary options for scripting out service deployments: Azure PowerShell, Azure CLI, and Infrastructure as Code templates. Azure PowerShell and the Azure CLI are command-line utilities that allow users to script their deployments with PowerShell or Bash. While these tools can be used to deploy services in Azure, the most common use for them is managing automated Infrastructure as Code deployments. Infrastructure as Code templates define the services being deployed and their desired settings. Terraform and Azure Resource Manager (ARM) are the most common Infrastructure as Code services that are used to automate Azure deployments. Building and deploying services with Terraform are outside of the scope for the DP-900 exam and will not be covered in this book. More information can be found at `www.terraform.io` if you would like to learn more about Terraform.

## Azure PowerShell

Azure PowerShell includes a powerful set of PowerShell cmdlets (pronounced command-lets) that can be used to manage and administer Azure services from a command line. Scripts developed with Azure PowerShell can be run in the Azure Portal through the *Azure Cloud Shell* or through the Windows PowerShell command prompt or Integrated Scripting Environment (ISE) on a local machine or VM. Keep in mind that developing and running Azure PowerShell scripts locally requires the *Azure Az* PowerShell module to be installed on the machine. Steps and considerations for installing the Azure Az PowerShell module can be found at `https://docs.microsoft.com/en-us/powershell/azure/install-az-ps?view=azps-6.3.0#installation`. This module comes preinstalled on the Azure Cloud Shell, allowing users to immediately use the Azure Az module cmdlets in PowerShell scripts.

The Azure Cloud Shell is a web-based interface that allows users to run PowerShell and Azure CLI scripts in the Azure Portal. You can access the Azure Cloud Shell by selecting the Cloud Shell icon in the upper-right corner of the Azure Portal. Figure 2.16 illustrates what this icon looks like in the Azure Portal.

**FIGURE 2.16** Azure Cloud Shell icon



Once the Azure Cloud Shell loads at the bottom of the screen, you will be able to develop and run Bash or PowerShell scripts to manage Azure services. Switch from Bash to PowerShell to run Azure PowerShell commands.

Relational databases can be easily deployed using an Azure PowerShell script. These scripts can define every option related to deploying a relational database, such as where it is deployed, the type of database, the administrator account username and password, network isolation settings, and the service tier. The following code snippet is an Azure PowerShell script that creates the following resources:

- A resource group to logically contain the logical server and its databases
- A logical server and an IP firewall rule that will open port 1433 on the logical server to a defined range of IP addresses
- The username and password for the server's administrator account
- An Azure SQL Database, its initial service tier, and the initial number of vCores it is allocated

```
<#
Sign into your Azure environment. Not
required if running this script in the Azure Cloud Shell
#>
Connect-AzAccount

<#
Set the ID for the Subscription this database
is being deployed to. Also not needed if running in the Azure Cloud Shell
#>
$SubscriptionId = "<Azure Subscription ID>"

# Set the resource group name and location for the logical server
$resourceGroupName = "sql001"
$location = "eastus2"

# Set an admin login and password for your server
$adminSqlLogin = "dp900admin"
$password = "<Admin Password>"

# Set a logical server name
$serverName = "dp900sql001sv"

# Set a database name
```

```
$databaseName = "dp900sql001db"

<#
The IP address range that you want to allow to
access your server. This is optional and can be
set after the deployment has finished.
#>
$startIp = "<First IP Address in Range>"
$endIp = "<Last IP Address in Range>"

# Set subscription
Set-AzContext -SubscriptionId $subscriptionId

# Create the resource group
$resourceGroup = New-AzResourceGroup -Name $resourceGroupName -Location `
$location

# Create the logical server
$server = New-AzSqlServer -ResourceGroupName $resourceGroupName `
 -ServerName $serverName `
 -Location $location `
 -SqlAdministratorCredentials $(New-Object -TypeName System.Management
.Automation.PSCredential
-ArgumentList $adminSqlLogin,
$(ConvertTo-SecureString -String $password -AsPlainText -Force))

<#
Create a server firewall rule that allows
access from the specified IP range
#>
$serverFirewallRule = New-AzSqlServerFirewallRule `
 -ResourceGroupName $resourceGroupName `
 -ServerName $serverName `
 -FirewallRuleName "AllowedIPs" -StartIpAddress $startIp -EndIpAddress `
$endIp

# Create a blank database that uses the General Purpose service tier
$database = New-AzSqlDatabase -ResourceGroupName $resourceGroupName `
 -ServerName $serverName `
 -DatabaseName $databaseName `
 -Edition "GeneralPurpose" `
 -Vcore 2
```

## Azure Command-Line Interface

The Azure CLI is a command-line tool that allows users to create and manage Azure resources. As with Azure PowerShell, scripts developed using the Azure CLI can be executed through the Azure Cloud Shell or through an interactive shell on a local machine or VM. Azure CLI commands can be run through a command prompt such as `cmd.exe` or through PowerShell on a Windows machine or a Bash shell in a Linux or macOS environment. Steps and considerations for installing the Azure CLI on a local machine or VM can be found at `https://docs.microsoft.com/en-us/cli/azure/install-azure-cli`.

The following code snippet is an Azure CLI script that performs the same actions as the previous Azure PowerShell script:

```
#!/bin/bash

# Set the subscription. Not required if being run in the Azure Cloud Shell
az account set—subscription <replace with your subscription name or id>

# Set the resource group name and location for your database
resourceGroupName=sql001
location=eastus2

# Set an admin login and password for the logical server adminlogin=dp900admin
password=<Admin Password>

# Set a logical server and database name
servername=dp900sql001sv
databasename=dp900sql001db

<#
The IP address range that you want to
allow to access your server. This is optional and
can be set after the deployment has finished.
#>
startip=<First IP Address in Range>
endip=<Last IP Address in Range>

# Create a resource group
az group create \
—name $resourceGroupName \
–location $location

# Create a logical server in the resource group
az sql server create \
–name $servername \
–resource-group $resourceGroupName \
–location $location \
–admin-user $adminlogin \
–admin-password $password

# Configure a firewall rule for the server
az sql server firewall-rule create \
–resource-group $resourceGroupName \
–server $servername \
 -n AllowYourIp \
–start-ip-address $startip \
–end-ip-address $endip

# Create a database in the server
az sql db create \
–resource-group $resourceGroupName \
–server $servername \
–name $databasename \
–edition GeneralPurpose \
–capacity 2
```

## Azure Resource Manager Templates

Before diving into how Azure Resource Manager (ARM) templates are defined, we first need to establish what ARM is. ARM is the deployment and management service that enables users to create, update, and delete resources in Azure. It receives, authenticates, and authorizes all requests made by APIs, the Azure Portal, Azure PowerShell, Azure CLI, or applications using one of the Azure SDKs.

ARM uses *resource providers* to know which Azure resources are involved in a request. Resource providers supply different resource types in Azure as well as all the configuration details that they require. One common resource provider is Microsoft.Sql, which includes the Azure SQL Database, Azure SQL MI, and Azure Synapse Analytics resource types. These resources can be specified using the syntax {resource provider}/{resource type}. Examples include Microsoft.Sql/servers or Microsoft.Sql/managedInstances. Resource providers are also the fundamental building blocks of ARM templates, as all other items in the template will be related to the configuration requirements of the resources defined in the template.

NOTE While many resource providers are registered to an Azure subscription by default, there are several that must be registered manually through the Azure Portal, Azure PowerShell, or Azure CLI. Steps to manually enable a resource provider can be found at `https://docs.microsoft.com/en-us/azure/azure-resource-manager/management/resource-providers-and-types#azure-portal`.

ARM templates are JSON files that define the resources and configuration requirements for a deployment to Azure. Templates are defined using a declarative syntax, meaning that they are written in a way that describes what resources are needed and each one's desired configuration without needing to worry about the programming commands that will create them. Resources defined in an ARM template can also have dependencies on other resources. Dependencies will prevent the template from attempting to deploying a resource if a resource it depends on is not available. Templates can then be deployed from Azure PowerShell and Azure CLI scripts, the Azure Portal, and tools like Azure DevOps that manage continuous integration and continuous development (CI/CD) pipelines.

You can think of an ARM template like a food order placed through an online delivery service. When you place an order, you declaratively list what items you want to eat. This may include appetizers, main dishes, side orders, and desserts, depending on what you want included in the order. The size of the order may also vary, depending on whether you are ordering just for yourself or also for other people. Certain items, such as a steak, also require you to state how you would like them to be cooked. Once the order is placed, the restaurant will handle the low-level details involved in preparing, cooking, and packaging the food.

NOTE Understanding the full scope of ARM templates and how they can be integrated into continuous integration and continuous deployment pipelines is outside of the scope for the DP-900 exam. If you would like to learn more about customizing ARM templates with parameters, information can be found at the following learning path: `https://docs.microsoft.com/en-us/learn/modules/create-azure-resource-manager-template-vs-code`.

### Defining an ARM Template

The following is a list of required and optional elements that make up an ARM template:

- *schema*—This is a required section that defines the location of the JSON schema file that describes the structure of the JSON data.
- *contentVersion*—This is a required section that defines the version of your template.
- *apiProfile*—This is an optional section that defines a collection of API versions for resource types.
- *parameters*—This is an optional section where you define values that are provided during deployment. Parameters are values that change depending on the environment the resources are being deployed to. These values can be provided by a parameter file, Azure PowerShell, or Azure CLI or in the Azure Portal.

- *variables*—This is an optional section where you define values that are reused in your template.
- *functions*—This is an optional section where you can define user-defined functions that simplify complicated expressions that may be used often in your template.
- *resources*—This is a required section where you define the resources you want to create or update in Azure.
- *output*—This is an optional section where you specify the values that will be returned at the end of the deployment.

The following is an example of an ARM template that will create an Azure SQL Database. The template definition includes the following elements:

- The logical server name.
- The username and password for the server's administrator account.
- An Azure SQL Database, its initial service tier, and its performance SKU.

```
{
  "$schema": "https://schema.management
.azure.com/schemas/2019-04-01/deploymentTemplate.json#",
  "contentVersion": "1.0.0.0",
  "parameters": {
    "serverName": {
      "type": "string",
      "defaultValue": "dp900sql001sv",
      "metadata": {
        "description": "The name of the SQL logical server."
      }
    },
    "sqlDBName": {
      "type": "string",
      "defaultValue": "dp900sql001db",
      "metadata": {
        "description": "The name of the SQL Database."
      }
    },
    "location": {
      "type": "string",
      "defaultValue": "eastus2",
      "metadata": {
        "description": "Location for all resources."
      }
    },
      "administratorLogin": {
        "type": "string",
        "metadata": {
          "description": "The administrator username of the SQL logical server."
        }
      },
      "administratorLoginPassword": {
        "type": "securestring",
        "metadata": {
          "description": "The administrator password of the SQL logical server."
        }
      }
    },
    "variables": {},
    "resources": [
      {
        "type": "Microsoft.Sql/servers",
        "apiVersion": "2020-02-02-preview",
        "name": "[parameters('serverName')]",
        "location": "[parameters('location')]",
        "properties": {
          "administratorLogin": "[parameters('administratorLogin')]",
          "administratorLoginPassword": "[parameters('administratorLoginPassword')]"
        },
        "resources": [
          {
            "type": "databases",
            "apiVersion": "2020-08-01-preview",
            "name": "[parameters('sqlDBName')]",
            "location": "[parameters('location')]",
            "sku": {
              "name": "GP_Gen5_2",
              "tier": "GeneralPurpose",
            },
            "dependsOn": [
              "[resourceId('Microsoft.Sql/servers',
                  concat(parameters('serverName')))]"
            ]
          }
        ]
      }
    ]
  }
```

This template can then be deployed through the Azure Portal, Azure PowerShell, or Azure CLI. The following is an example of an Azure PowerShell script that deploys the preceding ARM template to a new resource group. It also defines the administrator username and password for the logical server and passes the information to the template as it is being deployed. This script also assumes that the template is located in the same folder as the Azure PowerShell script with the name `azuredeploy.json`.

```
Connect-AzAccount

# Set an admin login and password for your server
$adminSqlLogin = "dp900admin"
$password = "<Admin Password>"

New-AzResourceGroup -Name sql001 -Location eastus
New-AzResourceGroupDeployment -ResourceGroupName `
arm-vscode -TemplateFile ./azuredeploy.json `
-administratorLogin $adminSqlLogin `
-administratorLoginPassword $password
```

TIP

There are several commonly used ARM templates available on the azure-quickstart-templates GitHub repository. These templates range from single resource deployments to multiple resource deployments for different application workloads. Feel free to reference these templates as you are building out your own automated deployments in Azure. You can find these templates at `https://github.com/Azure/azure-quickstart-templates`.

## Migrating to Azure SQL

There are a variety of methods available for migrating a database from an on-premises SQL Server instance to Azure SQL. Migrating a database to a SQL Server on Azure VM is relatively straightforward unless you are upgrading from an older version of SQL Server and need to update any deprecated features. The following migration options are commonly included:

- Taking a backup of the on-premises database and storing it in Azure Blob storage. Restore the database backup from Azure Blob storage to the SQL Server on Azure VM using RESTORE DATABASE FROM URL.
- If the on-premises instance is a primary in an Always On AG, add the SQL Server on Azure VM as a secondary. Once the data is synchronized to the SQL Server on Azure VM, perform a failover so that the SQL Server on Azure VM is the new primary.

- Configure transactional replication so that the on-premises SQL Server instance is a publisher and the SQL Server on Azure VM is a subscriber. Once the data is replicated to the SQL Server on Azure VM, update application connection strings and point users to the database in Azure.

Migrating to Azure SQL Database or Azure SQL MI requires more planning and consideration due to compatibility differences between on-premises SQL Server and PaaS Azure SQL. Even though Azure SQL MI is nearly 100 percent compatible with on-premises SQL Server, there are still some feature differences between the two that could cause migration issues. The same can be said about the differences between an on-premises instance of MySQL and PostgreSQL and Azure Database for MySQL and Azure Database for PostgreSQL. This is where a service such as the Azure Database Migration Service can provide data that makes the migration planning process much easier.

The *Azure Database Migration Service (DMS)* is a fully managed service that can be used to discover any potential compatibility issues and migrate the database once those issues are addressed. It uses the *Data Migration Assistant (DMA)* to detect compatibility issues and make recommendations on how to address them. DMA is also useful for migrations to a SQL Server on Azure VM by discovering compatibility issues between an older version of SQL Server and a newer version on the Azure VM. DMA can be used to assess versions of SQL Server ranging from SQL Server 2005 to the most up-to-date version. After addressing any compatibility issues, DMA can be used to migrate the database's schema to streamline data migration with DMS.

DMS can be used for offline and online migrations. *Offline* migrations refer to application downtime beginning as soon as the migration starts. Application cutover is a manual process and must be performed by the user. Offline migrations are available for migrations to Azure SQL Database, Azure SQL MI, SQL Server on Azure VM, Azure Cosmos DB, Azure Database for MySQL, and Azure Database for PostgreSQL. DMS can also limit downtime by handling the application cutover process through an *online* migration. Online migrations are only available for migrations to Azure SQL MI, Azure Cosmos DB, and Azure Database for PostgreSQL.

## Database Security

Database security is paramount for any RDBMS. For this reason, relational databases in Azure enforce database security through the following methods:

- Network isolation
- Access management
- Data encryption and obfuscation
- Security management

Each of these methods represents a different level of security for protecting data from nonauthorized access. While many of the tasks related to the different security layers are applied the same way across the different relational database offerings in Azure, there are some tasks that are handled differently from one database platform to another. For example, network isolation is implemented very differently on a SQL Server on Azure VM than it is on an Azure SQL Database.

Since network isolation was a core topic in the sections detailing the different relational database offerings in Azure, the following sections will focus on access management, data encryption and obfuscation, and security management capabilities.

## Access Management

Access management for relational databases in Azure is centered around the concept of *least-privilege*. This starts at the infrastructure level in Azure with role-based access controls (RBACs), allowing organizations to limit who can manage database operations that are handled in Azure such as changing maintenance windows and scaling compute resources users to only users who need this type of access. The next step is to limit database access to only the users that need access to it, also known as database authentication. Finally, users that can authenticate to a database will need to be granted varying levels of permission to the data and objects in the database, which should be set to the least amount of privilege needed by a user. This is known as a user's authorization level. The following sections explore these different levels of access management.

### Role-Based Access Control (RBAC)

Management operations for relational databases that are handled through Azure such as network isolation, scaling compute resources, and changing maintenance windows is controlled through RBAC. RBAC is an authorization system built on ARM that provides fine-grained access management of Azure resources to users and objects in Azure Active Directory. It is important to note that RBAC is decoupled from database-level security, so these roles do not affect database access.

Higher-level RBAC roles such as Owner and Contributor can be used to manage SQL resources but grant additional permissions that may not be necessary. There are built-in RBAC roles specific to Azure SQL that can be granted to Azure Active Directory accounts that eliminate the need for higher-level roles for managing Azure SQL resources. PaaS relational databases include the following built-in roles:

- *SQL DB Contributor*—Lets a user manage Azure SQL Databases but not access them. Also, this role does not allow users to manage the security-related policies or their associated logical servers.
- *SQL Managed Instance Contributor*—Lets a user manage Azure SQL MIs and required network configuration but not access them.
- *SQL Security Manager*—Lets a user manage the security-related policies of Azure SQL Databases and logical servers that manage databases but not access them.
- *SQL Server Contributor*—Lets a user manage Azure SQL Databases and their associated logical servers but not access them. Also, this role does not allow users to manage the security-related policies.

These roles do not apply to SQL Server on Azure VMs because the database engine is managed in the VM. However, there are VM-specific RBAC roles that can be used to manage the VM configuration. More on these and other built-in RBAC roles can be found at `https://docs.microsoft.com/en-us/azure/role-based-access-control/built-in-roles`.

## Authentication

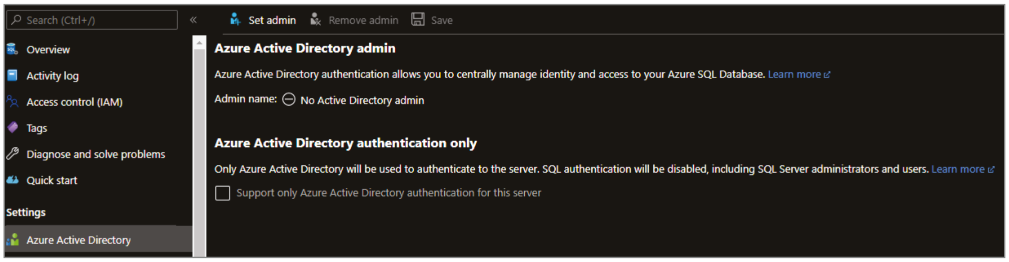
Authentication is the process of validating the identity of users trying to access a database. All versions of Azure SQL support two authentication methods: SQL authentication and Active Directory (AD).

*SQL authentication* involves storing SQL Server–specific login name and password information in the master database, or in the user database for database contained users. As a matter of fact, the administrator account that is defined when creating a SQL Server on Azure VM, Azure SQL MI, Azure SQL Database, or Azure Synapse Analytics dedicated SQL pool is an example of a SQL login. The administrator can also create additional SQL logins that other users or automation services such as the SQL Server Agent or Azure Data Factory can use to interact with the database.

*Active Directory authentication* involves adding a user or group stored in Windows AD or Azure AD (AAD) as a login or contained user in SQL. This is the preferred method of authentication as it is more secure than SQL authentication and is easier to manage. SQL Server on Azure VMs can use Windows AD logins for authentication if the VNet that contains the SQL Server on Azure VM is joined to a domain that has AD. As of the writing of this book, SQL Server on Azure VMs cannot use AAD users and groups for authentication. Azure SQL Database and Azure SQL MI, on the other hand, can use AAD objects. The following steps outline how to add an AAD user or group as an administrator for an Azure SQL Database logical server.

1. Log into `portal.azure.com` and search for *SQL servers* in the search bar at the top of the page. Click *SQL servers* to go to the SQL servers page in the Azure Portal. This page is the home of the logical servers for your Azure SQL Databases.
2. Click on the logical server that was created when you built an Azure SQL Database.
3. Click *Azure Active Directory* under Settings in the left-hand side panel. Click *Set admin* to add an AAD user or group as an administrator on the server. Figure 2.17 illustrates how this page will appear before clicking *Set admin*.

**FIGURE 2.17** Adding an AAD Administrator



**Azure Active Directory admin**

Azure Active Directory authentication allows you to centrally manage identity and access to your Azure SQL Database. Learn more

Admin name: No Active Directory admin

**Azure Active Directory authentication only**

Only Azure Active Directory will be used to authenticate to the server. SQL authentication will be disabled, including SQL Server administrators and users. Learn more

☐ Support only Azure Active Directory authentication for this server

4. Once you have added an account, click *Save* to save the account as the administrator.

Non-administrator AAD users and groups can also be added using T-SQL. To add additional AAD users and groups as database users, connect to the logical server using a management tool like SQL Server Management Studio (more on management tools later in this chapter) with a login that has permission to create users in the database. This can include the SQL authentication server administrator or the AAD server administrator. Once you're logged in, the following command can be used to add a contained user to a database.

```
CREATE USER [<AAD_User>] FROM EXTERNAL PROVIDER;
```

There are three methods available for using an AAD login to connect to a database. The correct choice depends on how an organization configures AAD. These methods are as follows:

- *Azure Active Directory—Integrated*: This method can be used if the user signed into the Windows machine that they are connecting to the database from with an AAD account.
- *Azure Active Directory—Password*: this method forces the user to enter the AAD login name and password to connect to the database.
- *Azure Active Directory—Universal with MFA*: This is an interactive method that uses multi-factor authentication (MFA) to provide additional access security for the database.

### Authorization

Authorization refers to the level of permissions a user has in the database. Some of these permissions include whether they can read or write data in different tables, execute stored procedures, and add or delete other users. Permissions are typically managed by database roles that include a predefined set of permissions. Database roles include fixed-database roles that are included in SQL Server and Azure SQL and user-defined database roles that are created by a database administrator.

NOTE There are several fixed-database roles available out of the box with any version of SQL Server or Azure SQL. An extensive list of the available fixed-database roles in Azure SQL can be found at `https://docs.microsoft.com/en-us/sql/relational-databases/security/authentication-access/database-level-roles?view=sql-server-ver15#fixed-database-roles`.

User permissions can also be managed by object-level permissions, such as granting or revoking the ability to select, update, or delete data in a specific table or view. Object-level permissions can also go as far as limiting which columns users have access to. An example of denying access to specific columns in a table with T-SQL would look like the following statement:

```
DENY SELECT ON <table_name>(<column_1, column_2>) TO User
```

There is also a special type of database authorization that limits access to specific rows in different tables. *Row-level security (RLS)* allows database administrators to control access to rows in a table based on the characteristics of the user running a query. This is implemented through user-defined table valued functions that block access to rows based on certain security predicates. RLS supports two types of security predicates to prevent user access to specific rows:

- Filter predicates that silently filter the rows available to read operations
- Block predicates that block write operations that violate the predicate

## Data Encryption and Obfuscation

Azure provides a variety of methods to protect data from malicious activity by encrypting data in-transit and at rest. These help to ensure that if a disk hosting a database, a data file, a database backup, or connections to a database becomes compromised, then the data is unreadable.

Azure SQL and open-source SQL databases in Azure use *Transport Layer Security (TLS)* to encrypt data in-transit. TLS encrypts data sent over the Internet to ensure that hackers are unable to see the data that is transmitted. Supported versions include 1.0, 1.1, and 1.2. Depending on application requirements, a minimum TLS version can be set so that application connections using the minimum allowed TLS version or higher can connect to that database.

Azure also encrypts data at rest by encrypting the disks that support the various database options. This ensures that if disks involved in hosting a database (e.g., data, log, and tempdb disks) are hacked, the data on those disks will be unreadable. Along with encrypting the physical disk, there are a few additional encryption measures that are native to SQL Server and Azure SQL that ensure a database is encrypted at rest. These are discussed further in the following sections.

### Transparent Data Encryption (TDE)

*Transparent Data Encryption (TDE)* is a SQL Server feature that encrypts all the data within a database at the page level. TDE is available for databases hosted in a SQL Server on Azure VM, Azure SQL Database, Azure SQL MI, and Azure Synapse Analytics dedicated SQL pool. Data is encrypted as it is written to the data page on disk and decrypted when the data page is read into memory. TDE also encrypts database backups since a backup operation is simply copying the data and log pages from the database.

Encryption with TDE is done by using a symmetric key called the Database Encryption Key (DEK). The DEK is managed by default by a service-managed certificate in Azure. Organizations can also use their own certificate, a method known as Bring Your Own Key (BYOK), to manage the DEK. Customer-managed certificates can be managed in Azure Key Vault.

NOTE

TDE is enabled by default for Azure SQL Database and Azure SQL MI and can be manually enabled for SQL Server databases that are hosted on a VM.

### Always Encrypted

In addition to encrypting entire databases at rest with TDE, SQL Server and Azure SQL allow organizations to encrypt individual columns in tables with *Always Encrypted*. This feature is designed to allow organizations to protect sensitive data such as credit card numbers or personally identifiable information (PII) stored in database tables. Always Encrypted allows client applications to encrypt data inside the application, never revealing the encryption keys to the database engine. This allows organizations to separate who can manage the data, like a database administrator, and who can read it.

Always Encrypted uses a column encryption key to encrypt the column data with either randomized encryption or deterministic encryption, and a master encryption key that encrypts the column encryption key. Neither of these are stored in the database engine and are instead stored in an external trusted key store such as Azure Key Vault. The only values of the two keys that are stored in the database engine are the encrypted values of the column encryption key and the information about the location of the master key.

Client applications accessing encrypted data must use an Always Encrypted client driver. The driver will be able to access the key store where the column and master encryption keys are located and will use them to decrypt the data as it is served to the application. Applications writing data to encrypted columns will also use the Always Encrypted client driver to ensure that data is encrypted as it is written. It is important to reiterate here that the data is never decrypted at the database engine, only at the application level.

### Dynamic Data Masking

*Dynamic data masking* limits the exposure of sensitive data to application users by obfuscating data in specific columns. Applications reading data from tables with masked columns do not need to be updated because dynamic data masking rules are applied in the query results, which does not change the data stored in the database. This means that users can view columns that are masked, but without seeing the actual data stored in the columns.

There are a variety of masking patterns that can be used to obfuscate column data. The following masking patterns are available for SQL Server and Azure SQL:

- *Default*—Fully masks the data in the column. Users will see XXXX for string values, 0 for numbers, and 01.01.1900 for date values.
- *Email*—Masks everything in an email address other than the first letter in the email and the suffix .com (e.g., jXXXX@XXXX.com).
- *Random*—Replaces numeric data with a random value from a specified range of values.
- *Custom*—Exposes the first and last digits of a piece of data and adds a custom padding string in the middle (e.g., 5XXX0).

These masking patterns can be enabled through the Azure Portal or T-SQL. There is also an additional pattern available through the Azure Portal.

Dynamic data masking is designed to limit data exposure to a set of predefined queries without any change needed to application code. However, it is important to note that the data that is masked is not encrypted and can be bypassed using inference or brute-force techniques. It is designed to be complementary to other security features such as TDE, Always Encrypted, and RLS.

## Security Management

Once data is secured through network isolation, access management, and data encryption and obfuscation techniques, it is important to make sure data security is maintained on an ongoing basis. The following methods are available through Azure and the database engine to manage database security.

### Auditing

Organizations enable auditing for Azure SQL to maintain regulatory compliance, understand database activities, and monitor databases for discrepancies that could indicate suspicious activity. SQL Server on Azure VM and Azure SQL MI use traditional SQL Server auditing through the database engine. This produces audit logs that contain predefined server-level or database-level events. Azure SQL Database and Azure Synapse Analytics dedicated SQL pools use Azure SQL Auditing to write audit logs to Azure Blob storage, Azure Log Analytics, or Azure Event Hubs. Azure SQL Auditing can be enabled through the Azure Portal.

### Azure Defender for SQL

Azure Defender provides several SQL security management capabilities. It includes functionality for monitoring and mitigating potential database vulnerabilities and detecting potentially malicious activity. It can be enabled through the Azure Portal at the Azure subscription level for all instances of Azure SQL in a subscription or at the server level for a single instance of Azure SQL. These security capabilities are covered by the following two tools that are packaged in the Azure Defender service: SQL Vulnerability Assessment and Advanced Threat Protection.

The SQL Vulnerability Assessment is a scanning service that provides insight into the state of your database's security. It also provides action items that a database administrator can take to resolve any found security issues. To catch security vulnerabilities in a database, the SQL Vulnerability Assessment employs several rules that are based on Microsoft best practices for database security. These rules cover database-level and server-level issues, such as firewall settings and excessive permissions for logins. The full list of rules that are used by the SQL Vulnerability Assessment can be found at `https://docs.microsoft.com/en-us/azure/azure-sql/database/sql-database-vulnerability-assessment-rules`.

Advanced Threat Protection is a tool that enables organizations to detect and respond to potentially malicious attempts to access a database. The tool will send alerts and recommended action items to users when it detects harmful database activities such as SQL injection, data exfiltration, anonymous logins, and brute force access. It is available for all versions of Azure SQL as well as Azure Synapse Analytics dedicated SQL pools.

## Common Connectivity Issues

There will be times when connectivity issues occur with a database. These issues can be related to network or firewall configuration, authentication timeouts, or transient fault errors related to Azure dynamically reconfiguring a database to meet heavy workloads. The following sections list common connectivity issues and how to troubleshoot them.

### Network-related or Instance-specific Issues

The "A network-related or instance-specific error occurred while establishing a connection to your server" error message indicates that an application cannot find the database server it is trying to connect to. The most common methods for troubleshooting this issue are as follows:

1. Making sure that TCP/IP is enabled as a client protocol on the application server. On servers that have SQL tools installed, such as a SQL Server on Azure VM, TCP/IP can be enabled by using the following steps in SQL Server Configuration Manager:
   a. Expand *SQL Server Native Client Configuration* and click on *Client Protocols*.
   b. Double-click *TCP/IP* and change *Enabled* from No to *Yes*.

   Application servers that do not have SQL tools installed can also be checked to see if TCP/IP is enabled by running the SQL Server Client Network utility (`cliconfig.exe`).
2. Make sure that the connection string specifies the right port (1433 by default) and is using the fully qualified server name. An example of a fully qualified logical server name for Azure SQL Database would be `dp900sql001.database.windows.net`.
3. Connection timeout can be the root cause for applications that are connecting over a slow network. This can be alleviated by increasing the connection timeout in SQL. The Microsoft recommended connection timeout is at least 30 seconds.

### Firewall-related Issues

The "Cannot connect to server due to firewall issues" error message indicates that the the client application's IP address is not whitelisted by the server-level or the database-level firewall. Add the IP address as a server-level or database-level firewall rule to alleviate this issue.

Keep in mind that if the database is hosted on an Azure SQL MI or is an Azure SQL Database that is using a private endpoint, then an application trying to communicate with the database will need to be able to communicate with the VNet the database is in. This would include the following applications:

- Applications that are hosted in the same VNet as the database.
- Applications that are hosted in a network that can communicate with the VNet hosting the database. This can be done through VNet peering, a VPN, or Azure ExpressRoute.

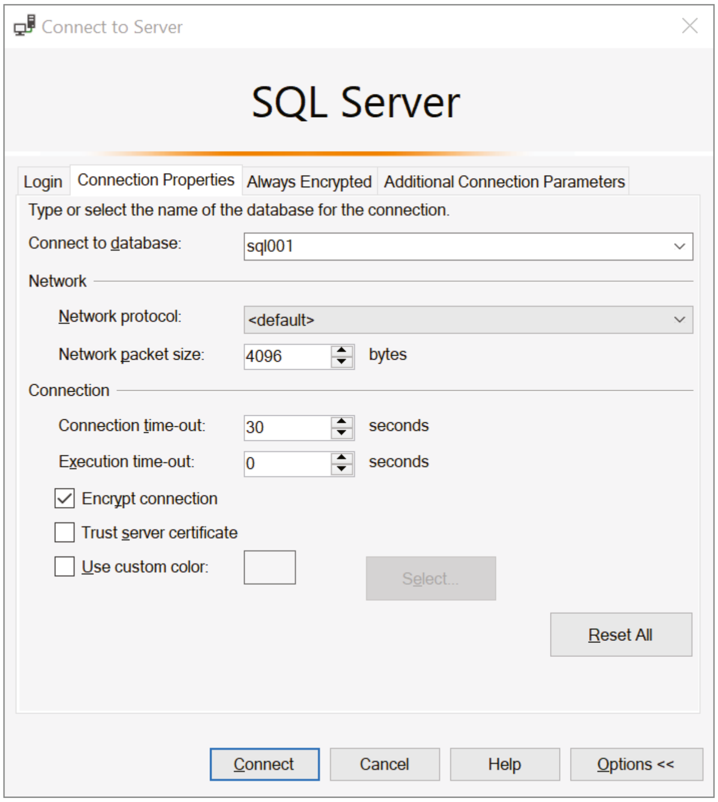
- Applications that are allowed to communicate with resources in a VNet hosting a database through an NSG or Azure Firewall. More on network security rules in NSGs can be found at `https://docs.microsoft.com/en-us/azure/virtual-network/network-security-groups-overview`.

## Log In Failure with a Database Contained User

The "Cannot open database "master" requested by the login. The login failed" error occurs because the account logging into the server does not have access to the master database. This is typical for database contained users that are trying to connect to the database with SQL Server Management Studio (SSMS). Use the following steps to resolve this issue:

1. When establishing a connection to a SQL instance in SSMS, click *Options* in the bottom left-hand corner of the Connect to Server page and select *Connection Properties*.
2. In the *Connect to database* field, type the name of the database the user is contained in and click *Connect*. Figure 2.18 is an example of the *Connection Properties* tab connecting to a specified user database.

**FIGURE 2.18** Connecting to a user database with SSMS



Type or select the name of the database for the connection.

Connect to database: sql001

Network protocol: <default>

Network packet size: 4096 bytes

Connection time-out: 30 seconds

Execution time-out: 0 seconds

☑ Encrypt connection

☐ Trust server certificate

☐ Use custom color:

### Transient Fault Errors

Transient fault errors occur when Azure dynamically reconfigures the infrastructure on which the database is hosted. These can include planned events such as database upgrades and unplanned events such as load balancing. Reconfiguration events that cause transient fault errors are typically short-lived and last less than 60 seconds. However, this can still cause problems since applications connecting to databases during this time may experience some connectivity issues. For this reason, applications should be built with retry logic to repeat a transaction if it fails due to a transient error. Transient errors are raised by the throw of a *SqlException* and are identified as one of a few error codes. This allows error handling logic to include a retry policy for exceptions that include a transient error code. The full list of transient error codes can be found at `https://docs.microsoft.com/en-us/azure/azure-sql/database/troubleshoot-common-errors-issues#transient-fault-error-messages-40197-40613-and-others`.

NOTE More information on transient fault errors and other common connectivity issues can be found at `https://docs.microsoft.com/en-us/azure/azure-sql/database/troubleshoot-common-errors-issues`.

## Management Tools

In previous sections, we established that tools like the Azure Portal and Azure PowerShell are powerful mechanisms for managing relational database deployments in Azure. However, there are other tools that developers use to write, test, and optimize queries before adding them to applications. These tools are also used by database administrators to perform tasks such as managing table design, indexes, and user permissions. The following sections provide a brief overview of the three most popular database management tools.

### SQL Server Management Studio

SQL Server Management Studio, or SSMS for short, has been used by database administrators and developers for years. It can connect to any type of SQL Server–based infrastructure, including SQL Server, Azure SQL, and Azure Synapse Analytics dedicated SQL pools. Once connected to a database, SSMS can be used to administer and develop all components of SQL, including the following tasks:

- Building and managing database objects such as tables, stored procedures, functions, and triggers
- Developing and optimizing queries
- Managing security operations
- Performing database backup and restore operations
- Building HADR solutions such as an Always On availability group

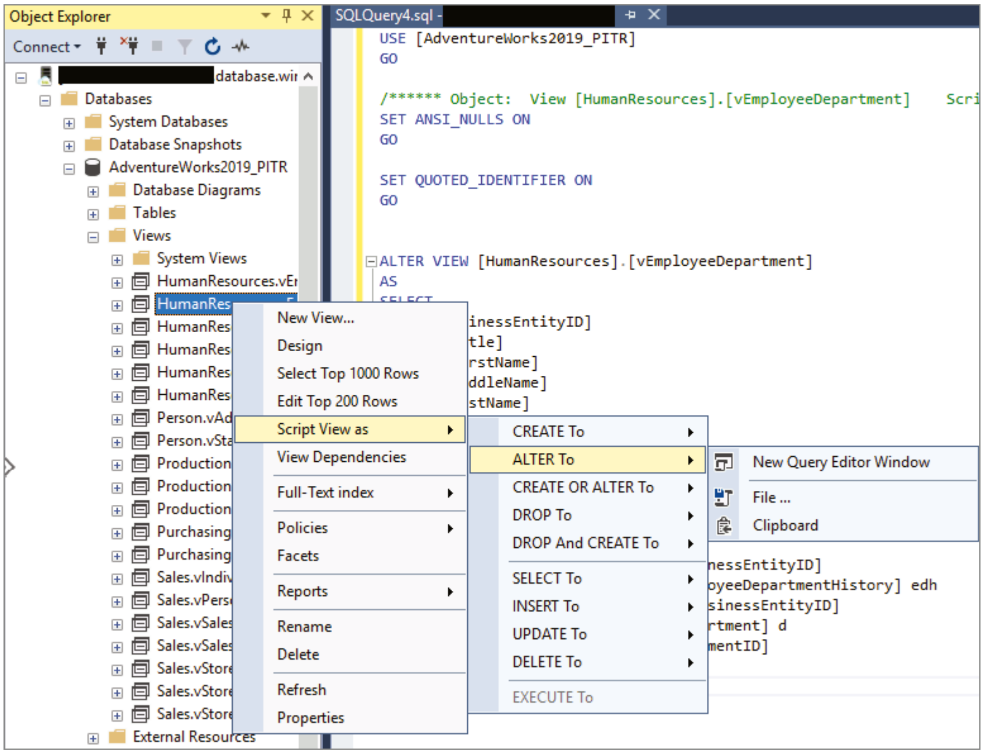
Many of these activities can be done through either the GUI or a T-SQL script that is written and executed in SSMS's Query Editor.

**NOTE** You can download the latest edition of SSMS at `https://docs.microsoft.com/en-us/sql/ssms/download-sql-server-management-studio-ssms?view=sql-server-ver15#download-ssms`.

After opening SSMS, you will be prompted to connect to a database server. Connections to Azure SQL Database and Azure SQL MI will use the endpoint created for the server. This can be found in the *Overview* page listed next to *Server Name for Azure SQL Database and Host for Azure SQL MI*. Once you have entered the server name, you will need to choose which type of authentication you will be using and enter the credentials. Remember that if you are logging in with a user that does not have access to the master database, you will need to specify the database you are connecting to in the *Connection Properties*.

Once connected, users can begin writing queries by clicking *New Query* in the top ribbon. This will open a new page in the Query Editor, with results being displayed at the bottom of the Query Editor after a query is run. SSMS also enables users to script out any object in a database by right-clicking on them in the Object Explorer, hovering the mouse over *Script <object> as*, and choosing one of the "script as" options. Figure 2.19 illustrates an example of how to script out an ALTER VIEW statement for an existing database view in SSMS. This example opens the script in a new Query Editor window.

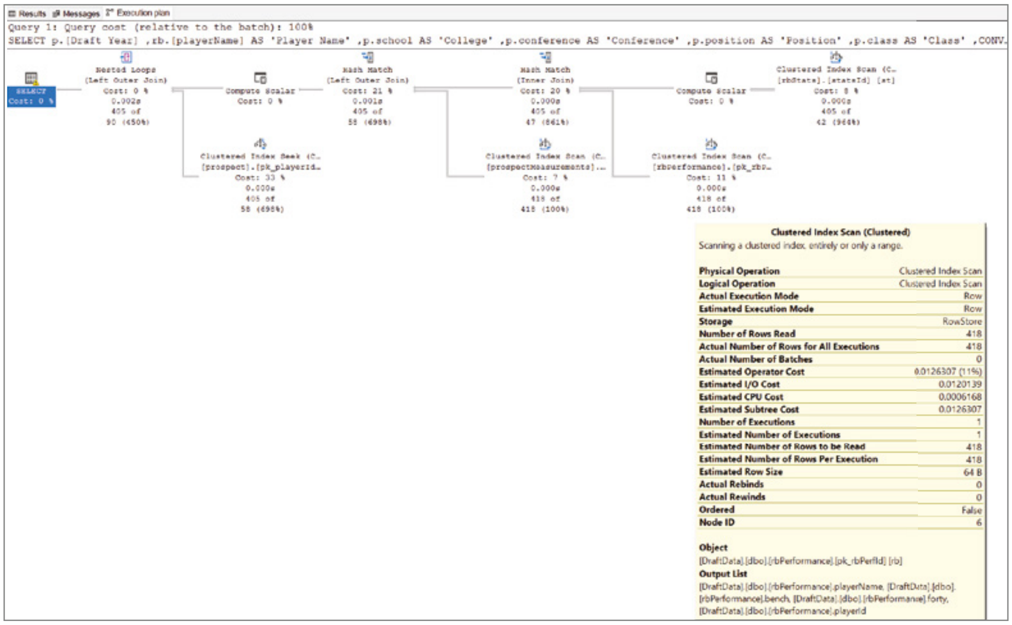
**FIGURE 2.19** Script View as ALTER To statement



```
USE [AdventureWorks2019_PITR]
GO

/****** Object:  View [HumanResources].[vEmployeeDepartment]    Scri
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

ALTER VIEW [HumanResources].[vEmployeeDepartment]
AS
SELECT
                  inessEntityID]
                  tle]
                  rstName]
                  ddleName]
                  stName]

                                      nessEntityID]
                                      oyeeDepartmentHistory] edh
                                      sinessEntityID]
                                      rtment] d
                                      mentID]
```

There are several administrative features that are native to SSMS, including the ability to optimize a query based on its execution plan. An *execution plan* is a graphical interpretation of the steps the database engine's query optimizer takes to execute a query. It is read right to left and displays metrics for each step, including the operation that was performed. SSMS will also display resource usage for a step if you hover your mouse over it. Figure 2.20 illustrates an example of an execution plan and includes the pop-up infographic for one of the steps.

**FIGURE 2.20** SSMS execution plan



```
Query 1: Query cost (relative to the batch): 100%
SELECT p.[Draft Year] ,rb.[playerName] AS 'Player Name' ,p.school AS 'College' ,p.conference AS 'Conference' ,p.position AS 'Position' ,p.class AS 'Class' ,CONV…
```

**Clustered Index Scan (Clustered)**

Scanning a clustered index, entirely or only a range.

| | |
|---|---|
| **Physical Operation** | Clustered Index Scan |
| **Logical Operation** | Clustered Index Scan |
| **Actual Execution Mode** | Row |
| **Estimated Execution Mode** | Row |
| **Storage** | RowStore |
| **Number of Rows Read** | 418 |
| **Actual Number of Rows for All Executions** | 418 |
| **Actual Number of Batches** | 0 |
| **Estimated Operator Cost** | 0.0126307 (11%) |
| **Estimated I/O Cost** | 0.0120139 |
| **Estimated CPU Cost** | 0.0006168 |
| **Estimated Subtree Cost** | 0.0126307 |
| **Number of Executions** | 1 |
| **Estimated Number of Executions** | 1 |
| **Estimated Number of Rows to be Read** | 418 |
| **Estimated Number of Rows Per Execution** | 418 |
| **Estimated Row Size** | 64 B |
| **Actual Rebinds** | 0 |
| **Actual Rewinds** | 0 |
| **Ordered** | False |
| **Node ID** | 6 |

**Object**
[DraftData].[dbo].[rbPerformance].[pk_rbPerfId] [rb]
**Output List**
[DraftData].[dbo].[rbPerformance].playerName, [DraftData].[dbo].[rbPerformance].bench, [DraftData].[dbo].[rbPerformanse].forty, [DraftData].[dbo].[rbPerformance].playerId

## Azure Data Studio

Azure Data Studio is an open-source database management tool that can be used on a Windows, macOS, or Linux machine. Like SSMS, it can connect to SQL Server, Azure SQL, and Azure Synapse Analytics dedicated SQL pools. Azure Data Studio provides a modern developer experience with features such as IntelliSense, source control integration, and an integrated terminal. Not only does it display results for queries, but it also comes with built-in charting to allow users to visualize trends and data skew.

NOTE You can download the latest edition of Azure Data Studio at `https://docs.microsoft.com/en-us/sql/azure-data-studio/download-azure-data-studio?view=sql-server-ver15`.

Once it's launched, you can connect to a database with Azure Data Studio by clicking *New Connection*. This will open a new window to the right where you can add the server name, authentication type, and credentials. There is also an option to change the database from <Default> to a user database if you are logging in with a database contained user.

After you click *Connect*, Azure Data Studio will display the type (e.g., Azure SQL Database, Azure SQL MI, etc.), SQL Server version, and the databases that are hosted on the server. Users can then choose *New Query* or *New Notebook* to begin developing code. Clicking *New Query* opens a query window that provides a similar experience to the SSMS Query Editor. Clicking *New Notebook* opens a Jupyter Notebook that allows users to write queries using SQL, Python, Julia, R, Scala, and PowerShell code.

## Sqlcmd

Sqlcmd is a command-line utility that can be used to connect and query databases hosted in SQL Server, Azure SQL, and Azure Synapse Analytics dedicated SQL pools. The utility allows users to enter T-SQL statements or run script files through a command prompt. It includes several built-in switches that can be used for tasks such as authenticating to a database, running a query from a file, and configuring what information is returned with a query. Some of the most common sqlcmd switches are listed in Table 2.8.

**TABLE 2.8** Common sqlcmd switches

| Switch | Definition |
|---|---|
| `-d` | Database Name |
| `-E` | Use Trusted Connection |
| `-g` | Enable Column Encryption |
| `-G` | Use AAD Authentication |
| `-i` | Input File |
| `-K` | Set Application Intent (useful for read-only workloads) |
| `-l` | Login Timeout |
| `-m` | Error Level |
| `-N` | Encrypt Connection |
| `-o` | Output File |
| `-P` | Password |
| `-S` | Server Name |

| Switch | Definition |
|---|---|
| `-t` | Query Timeout |
| `-U` | Username |
| `-V` | Error Severity Level |
| `-z` | New Password |
| `-Z` | New Password and exit |

NOTE You can download the latest edition of sqlcmd at `https://docs.microsoft.com/en-us/sql/tools/sqlcmd-utility?view=sql-server-ver15#download-the-latest-version-of-sqlcmd-utility`.

To use sqlcmd, open a command prompt and type **`sqlcmd`** followed by the server information and authentication details. The following is an example of a sqlcmd command connecting to an Azure SQL Database:

```
sqlcmd -S <server_name>.database.windows.net
-d <database_name> -U <user_name> -P <password>
```

Running a query from the command prompt with sqlcmd can be easily performed by entering the `sqlcmd` command followed by the query. The following is an example of a query in sqlcmd that returns every row in a table:

```
sqlcmd
USE <database_name>;
GO
SELECT * FROM <table_name>;
GO
```

## Query Techniques for SQL

As mentioned in Chapter 1, SQL is the development language used to build, access, and manipulate relational databases. The American National Standards Institute (ANSI) and the International Organization for Standardization (ISO) recognizes SQL as a standard language. While ANSI SQL is the standard that all major relational database vendors adhere to, most of them extend the language with functionality custom to their relational database products. For example, T-SQL is the Microsoft extension of ANSI SQL that is native to SQL Server, Azure SQL, Azure SQL Database, Azure SQL MI, and Azure Synapse Analytics.

## DDL vs. DML Commands

Standard ANSI SQL commands can be broken down into two primary categories: Data Definition Language (DDL) and Data Manipulation Language (DML). DDL commands are used to define relational database objects such as databases, tables, views, stored procedures, and triggers. DML commands are used to manipulate data stored in a relational database. The following sections describe common commands and statement structure used by these categories.

NOTE Chapter 1 describes two additional SQL categories with DDL and DML commands that are specific to T-SQL. These include Data Control Language (DCL) commands that can be used to manage permissions and Transaction Control Language (TCL) commands that are used to control transaction execution.

### Data Definition Language (DDL)

DDL statements are used to define database objects. They can be used to create new objects, modify existing ones, or remove objects that are no longer required. DDL statements start with a command that indicates which of these actions the statement is performing. Table 2.9 includes a list of these commands and the common statement structures associated with each of them.

**TABLE 2.9** DDL commands

| Command | Description | Statement Structure |
|---|---|---|
| CREATE | Create a new object in a database. | `CREATE TABLE <table name>`<br>`(`<br>`<list of columns>`<br>`)` |
| ALTER | Modify the structure of an existing database object. | `ALTER TABLE <table name>`<br>`ADD COLUMN <column name>` |
| DROP | Remove an object from a database. | `DROP TABLE <table name>` |
| RENAME | Rename an existing object. | `EXEC sp_rename <old name>, <new name>` |
| TRUNCATE | Remove all rows from a table. | `TRUNCATE TABLE <table name>` |

NOTE SQL Server and Azure SQL do not have a RENAME command. Instead, user-defined objects can be renamed using the `sp_rename` system stored procedure.

Along with the database objects, DDL statements can also define what type of data can be stored in object columns. *Data types* specify the type of data a column can store, which also defines what kind of actions can be performed on that column. For example, columns defined as numeric data types can be aggregated in ways that a string data type cannot. Table 2.10 includes a list of some of the most popular SQL data types and their descriptions.

**TABLE 2.10** Common SQL data types

| Data Type | Description |
|---|---|
| INT | Used to define numeric data that rounds to a whole number. |
| DECIMAL(*p, s*) | Used to define numeric data that has fixed precision (*p*) and scale (*s*). |
| FLOAT(*n*) | Used to define numeric data that has approximate, or floating, decimal places. |
| BIT | Used to define numeric data that can take a value of 1, 0, or NULL. |
| DATE | Used to define a date. |
| DATETIME | Used to define a date that is combined with a time of day. |
| VARCHAR(*n*) | Used to define string data that has variable size. *n* is used to define the number of characters that can be stored. |
| NVARCHAR(*n*) | Unicode version of the VARCHAR(*n*) data type. The storage size is two times the number of characters. |
| CHAR(*n*) | Used to define string data that has a fixed size. *n* is used to define the number of characters that can be stored. |
| NCHAR(*n*) | Unicode version of the CHAR(*n*) data type. The storage size is two times the number of characters. |

Data types are not the only way DDL commands can define table data. Constraints are used in conjunction with data types to limit the type of data that can be stored in a column. If a statement inserting or updating data violates the constraint, then the action is immediately canceled. Table 2.11 includes a list of some of the most used constraints.

**TABLE 2.11** Common SQL constraints

| Command | Description |
|---|---|
| NOT NULL | Ensures that a column has a value for every row. |
| UNIQUE | Ensures that all values in a column are different. |
| PRIMARY KEY | Uniquely identifies each row in a table. Also uses NOT NULL and UNIQUE constraints to ensure there are unique values for every row. |
| FOREIGN KEY | Used to create relationships with other tables. Prevents any action from breaking a relationship. |
| CHECK | Used to specify what data values are acceptable in one or more columns. |
| DEFAULT | Sets a default value for a column if a value is not specified when new data is inserted. |
| INDEXES | Used to enhance the performance of queries. Depending on the index type, they can physically order data in an object or provide pointers to the physical location of data. |

Now that we have discussed DDL commands, data types, and constraints, let's explore how these can be used to construct a DDL statement. The following statement creates a table called DimProductCategory.

```
CREATE TABLE [dbo].[DimProductCategory](
.....[ProductCategoryKey] [int] IDENTITY(1,1) NOT NULL,
.....[ProductCategoryAlternateKey] [int] NULL,
.....[EnglishProductCategoryName] [nvarchar](50) NOT NULL,
.....[SpanishProductCategoryName] [nvarchar](50) NOT NULL,
.....[FrenchProductCategoryName] [nvarchar](50) NOT NULL,
CONSTRAINT [PK_DimProductCategory_ProductCategoryKey] PRIMARY KEY CLUSTERED
(
.....[ProductCategoryKey] ASC
)ON [PRIMARY],
CONSTRAINT [AK_DimProductCategory_ProductCategoryAlternateKey]
 UNIQUE NONCLUSTERED
(
.....[ProductCategoryAlternateKey] ASC
)ON [PRIMARY]
)
GO
```

The statement begins by declaring that it is going to create a new table in the database. Columns, their data types, and constraints are then defined between the open and close parentheses. Indexes defined in the table statement also include the columns on which they are based and the ascending or descending sort direction for the column.

Note that the ProductCategoryKey column definition also includes the IDENTITY key word. This property is used to ensure that primary key or unique constraint columns have unique values generated for every new row that is inserted. Unless this property is turned off by using the SET IDENTITY_INSERT ON command at the beginning of a transaction, identity columns do not allow user modifications. Instead, values for identity columns are generated based on the *seed* and *increment* arguments defined in the `CREATE TABLE` statement. For example, the first row inserted in the DimProductCategory table will generate the value 1 in the ProductCategoryKey column since the *seed* argument is set to 1. The second argument represents the incremental value that is added to the previous row that was loaded. In this case, the second row inserted will generate the value 2 in the ProductCategoryKey column since the *increment* value is set to 1 and the first row's ProductCategoryKey column equals 1.

## Data Manipulation Language (DML)

DML statements are used to manipulate data stored in a database. They can be used to retrieve and aggregate data for analysis, insert new rows, or edit existing rows. Table 2.12 lists the four main DML commands and the common statement structures associated with each of them.

**TABLE 2.12** DML commands

| Command | Description | Statement Structure |
|---|---|---|
| SELECT | Read rows from a table or view | `SELECT` *`<list of columns>`* `FROM` *`<table name>`* `WHERE` *`<filter condition>`* `GROUP BY` *`<group by expression>`* `HAVING` *`<search condition>`* `ORDER BY` *`<columns to sort by>`* |
| INSERT | Insert new rows into a table | `INSERT INTO` *`<table name>`* `(` *`<list of columns>`* `)` `VALUES` `(` *`<values to insert>`* `)` |

**TABLE 2.12** DML commands *(continued)*

| Command | Description | Statement Structure |
|---|---|---|
| UPDATE | Update existing rows | `UPDATE <table name>`<br>`SET <column> = <new value>`<br>`WHERE <filter condition>` |
| DELETE | Remove existing rows | `DELETE FROM <table name>`<br>`WHERE <filter condition>` |

Select statements are often more sophisticated than the example structure illustrated in Table 2.12. Queries can retrieve data from multiple tables, convert column data types, and perform aggregations. The UNION, EXCEPT, and INTERSECT operators can also be used to combine or contrast results from multiple queries into one result set. There will be more sophisticated query examples in the following sections, but it is important to note that processing order of operations in a select statement does not match the order they are written. This order, also known as the *logical processing order*, determines when the results from one step are made available to subsequent steps. The logical processing order is defined as follows:

1. FROM
2. ON
3. JOIN
4. WHERE
5. GROUP BY
6. WITH CUBE or WITH ROLLUP
7. HAVING
8. SELECT
9. DISTINCT
10. ORDER BY
11. TOP

More information on the structure of a T-SQL select statement can be found at `https://docs.microsoft.com/en-US/sql/t-sql/queries/select-transact-sql?view=sql-server-ver15`.

## Query Relational Data in Azure SQL, MySQL, MariaDB, and PostgreSQL

While most RDBMSs implementations of SQL use the same core functionality, there are some subtle differences. The following sections explore the syntax used to query data in Azure SQL, Azure Database for MySQL, Azure Database for MariaDB, and Azure Database for PostgreSQL and highlights some of the key differences.

### Querying Azure SQL with T-SQL

The first set of queries discussed are constructed using T-SQL. As mentioned previously, T-SQL is the Microsoft extension of ANSI SQL used to communicate with a SQL Server–based relational database. All the examples in this section can be used to query tables in the AdventureWorksDW2019 database. Use the following link to download a backup of the database: `https://docs.microsoft.com/en-us/sql/samples/adventureworks-install-configure?view=sql-server-ver15&tabs=ssms#download-backup-files`. The link also provides instructions on how to restore the database to an instance of SQL Server or Azure SQL.

Retrieving data from a relational database all starts with a select command. The following is an example of a select statement that returns all data from every column in a single table:

```
SELECT *
FROM [dbo].[FactInternetSales]
```

The asterisk (*) symbol is a wildcard character that indicates "all." In this case, the * is used to represent every column in the FactInternetSales table. While this saves users time when writing queries with long column lists, it can result in poor query performance by returning more data than what is required. Also, applications that use `SELECT *` statements are liable to break when new columns are added to the table or view being queried. For these reasons, it's always better to explicitly list the columns needed in a select statement.

Queries are often written to return a filtered list of data. The following is an example that returns sales information only for products that cost more than $1,000. The result set is also sorted by sales amount in descending order.

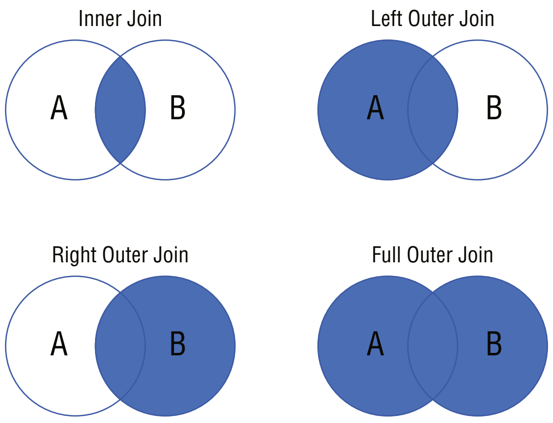
```
SELECT ProductKey
    ,ProductStandardCost
    ,TotalProductCost
    ,SalesAmount
    ,OrderDate
FROM [dbo].[FactInternetSales]
WHERE ProductStandardCost > 1000
ORDER BY SalesAmount DESC
```

Rarely do applications and reports use data from only one table. Instead, applications querying relational databases will often build result sets from two or more tables with the same select statement. Queries can do this by using a *join* operation. Join operations leverage the logical relationships between tables to build rows of data with columns from different tables. There are different types of join operations available that can return different combinations of data. The following is a list of the four most common join types and their T-SQL implementations:

- Inner joins retrieve data from both tables that meets the join condition. Inner joins can be defined with the INNER JOIN or JOIN expressions.
- Left outer joins retrieve all data from the table on the left-hand side of the join condition and data from the right table that meets the join condition. Left outer joins can be defined with the LEFT OUTER JOIN or LEFT JOIN expressions.
- Right outer joins retrieve all data from the table on the right-hand side of the join condition, and data from the left table that meets the join condition. Right outer joins can be defined with the RIGHT OUTER JOIN or RIGHT JOIN expressions.
- Full outer joins retrieve all data from both the left and right tables. Full outer joins can be defined with the FULL OUTER JOIN or FULL JOIN expressions.

Figure 2.21 illustrates how the different join types retrieve data from two tables (represented as table A and table B).

**FIGURE 2.21** Types of SQL joins



In addition to these join types, SQL Server–based database engines also enable users to develop queries using *cross joins*. Cross joins are special types of joins that return the Cartesian product of rows from both tables. Cross joins can be defined with the CROSS JOIN expression.

The following query builds on the previous example with added data from the DimProduct table. Since it uses a `JOIN` command without any additional adjectives, the query will perform an inner join, only returning data from both tables that meet the join condition defined in the ON clause.

```
SELECT P.EnglishProductName
     ,FIS.ProductKey
     ,FIS.ProductStandardCost
     ,FIS.TotalProductCost
     ,FIS.SalesAmount
     ,FIS.OrderDate
FROM [dbo].[FactInternetSales] AS FIS
    JOIN [dbo].[DimProduct] AS P
        ON FIS.ProductKey = P.ProductKey
WHERE ProductStandardCost > 1000
ORDER BY SalesAmount DESC
```

Note that the query uses the AS command in the FROM clause to give each table a short form alias. This alias can be used to specify which tables the selected columns are in and how the join condition is defined. Aliases can also be given to columns, allowing users to give names to columns that are the result of aggregations.

SQL provides several built-in functions that can be used to infer insights out of relational data. Built-in functions can be categorized based on the actions they perform on data. For example, aggregate functions such as SUM(), MAX(), and MIN() perform calculations on a set of values and return a single value. They can be used in combination with the GROUP BY clause to calculate aggregations on categories of rows.

The following query revises the previous one so that it returns the total quantity sold and the total sales dollars for products that cost more than $1,000. It also groups the sales totals by product name and monthly sales per year.

```
SELECT P.EnglishProductName
     ,SUM(FIS.OrderQuantity) AS TotalQuantity
     ,SUM(FIS.SalesAmount) AS TotalSales
     ,MONTH(FIS.OrderDate) AS [Month]
     ,YEAR(FIS.OrderDate) AS [Year]
FROM [dbo].[FactInternetSales] AS FIS
    JOIN [dbo].[DimProduct] AS P
        ON FIS.ProductKey = P.ProductKey
WHERE ProductStandardCost > 1000
      AND YEAR(FIS.OrderDate) > 2010
GROUP BY P.EnglishProductName,
         MONTH(FIS.OrderDate),
         YEAR(FIS.OrderDate)
ORDER BY [Year], TotalSales DESC
```

You may be wondering why the WHERE and GROUP BY clauses are not using the column aliases that were defined at the beginning of the statement. This is due to the T-SQL logical processing order that was discussed previously in this chapter. Since the WHERE and GROUP BY clauses are processed by the database engine before the SELECT is, these clauses do not know how to resolve column aliases.

The final T-SQL example in this section describes how to limit the result set to the first 10 rows the query returns. This is one key difference between T-SQL and other versions of SQL, as T-SQL uses the TOP command and other versions use LIMIT. We will demonstrate how other relational database platforms implement the LIMIT command.

```
SELECT TOP(10) P.EnglishProductName
     ,SUM(FIS.OrderQuantity) AS TotalQuantity
     ,SUM(FIS.SalesAmount) AS TotalSales
FROM [dbo].[FactInternetSales] AS FIS
    JOIN [dbo].[DimProduct] AS P
        ON FIS.ProductKey = P.ProductKey
WHERE ProductStandardCost > 1000
GROUP BY P.EnglishProductName
ORDER BY TotalSales DESC
```

## Querying MySQL, MariaDB, and PostgreSQL

Queries written to interact with MySQL, MariaDB, and PostgreSQL databases are very similar to ones written in T-SQL. The following example is nearly identical to the previous T-SQL query, with one key difference.

```
SELECT P.EnglishProductName
     ,SUM(FIS.OrderQuantity) AS TotalQuantity
     ,SUM(FIS.SalesAmount) AS TotalSales
FROM [dbo].[FactInternetSales] AS FIS
    JOIN [dbo].[DimProduct] AS P
        ON FIS.ProductKey = P.ProductKey
WHERE ProductStandardCost > 1000
GROUP BY P.EnglishProductName
ORDER BY TotalSales DESC
LIMIT 10
```

The SQL dialects used by MySQL, MariaDB, and PostgreSQL do not use the TOP(*n*) command to retrieve the first *n* number of rows that are returned by a query. Instead, these dialects use the LIMIT command to limit the number of rows returned.

Keep in mind that queries written to retrieve and manipulate data stored in one of these database engines will need to be done from a tool that can connect to them. MySQL Workbench is a graphical tool that is like SSMS that can be used to connect to MySQL and MariaDB databases. Queries developed for PostgreSQL databases can be done using the graphical tool pgAdmin.

## Summary

The "relational data on Azure" objective of the DP-900 exam focuses on building a foundational understanding of common relational database workloads and database structures. It focuses on the different types of relational database offerings in Azure, along with deployment, security, and development considerations for them.

This chapter covered the following concepts:

**Describe relational data workloads.** Relational data workloads can be split between transactional and analytical. Transactional, or OLTP, workloads store interactions that are related to an organization's activities, such as retail purchases. Databases such as SQL Server and Azure SQL Database include mechanisms for managing concurrent transactions to maintain ACID compliancy. Unlike OLTP workloads that are focused on optimizing database writes, analytical workloads are optimized for read-heavy applications. Analytical databases are flattened for this reason, so that users reading data do not have to write overly complex queries to query data.

**Describe relational Azure data services.** There are several relational database options on Azure that are designed to meet any organizational need. The Azure SQL portfolio of products include relational database offerings that use the Microsoft SQL Server database engine. These include SQL Server on Azure Virtual Machine, Azure SQL Managed Instance, and Azure SQL Database. Organizations needing horizontal scale for data warehouse and big data analytics workloads can use an Azure Synapse Analytics dedicated SQL pool. Azure Database for PostgreSQL, Azure Database for MariaDB, and Azure Database for MySQL enable organizations to offload infrastructure and management of their on-premises open-source relational database footprint to Azure.

**Describe common management tasks for relational databases in Azure.** Relational databases hosted in Azure remove tedious activities associated with managing infrastructure, allowing organizations to spend more time on building solutions that provide valuable insights. However, there are still several management activities that need to be maintained by an administrator, such as automating environment deployments and managing security. For this reason, Azure provides various options for organizations to automated database deployments that are both flexible and highly scalable. Security is also provided at multiple layers in both Azure and in the database engine. These can be categorized by network isolation, access management, data encryption and obfuscation, and security management. There are also several tools provided by Microsoft that allow database administrators and developers to easily perform the activities required to maintain a highly performant relational database solution.

**Describe common query techniques.** The Structured Query Language, or SQL for short, is an ANSI/ISO-compliant development language that is used to interact with relational data. SQL commands can be categorized into four different types: Data Definition Language (DDL), Data Manipulation Language (DML), Data Control Language (DCL), and

Transaction Control Language (TCL). Of these, DDL and DML statements are the most important to understand for the DP-900 exam. While ANSI SQL is the standard that all major RDBMSs use, most of them extend the language with some custom functionality. For example, Transact-SQL (T-SQL) is the implementation of SQL that is used by SQL Server–based relational databases.

## Exam Essentials

**Identify the right data offering for a relational workload.** For this topic, be sure to know when to use an SMP database such as Azure SQL, a MPP database such as Azure Synapse Analytics dedicated SQL pools, and an OLAP tool such as Power BI. SMP databases are typically used for transactional workloads and smaller data warehouses. MPP databases are used for large analytical workloads. OLAP tools are used as data marts or the semantic layer for analytical workloads and include predefined aggregations that are ready to be reported against.

**Describe relational data structures.** Be able to identify relational database structures such as tables, views, indexes, stored procedures, functions, and triggers. Tables are the basic storage object for a relational database and store elements of data as rows with one or more attributes stored as columns. Views are virtual objects whose contents are defined by a query returning data from one or more tables. Indexes optimize queries by sorting data physically and providing pointers to where data is stored. Stored procedures and functions encapsulate regularly used T-SQL code to minimize the code footprint needed for applications. Triggers are code blocks that are executed in response to DML, DDL, or login-based events.

**Describe IaaS and PaaS Azure SQL services.** Azure has multiple options for hosting relational databases with different levels of abstraction and administrative effort. Of these options, SQL Server on Azure VMs is most like an on-premises SQL Server instance as it gives organizations the most control over the OS and the database engine. This is an example of an Infrastructure as a Service, or IaaS, offering. Organizations looking to offload the management of the OS and features of the database engine can take advantage of Platform as a Service, or PaaS, offerings such as Azure SQL Managed Instance and Azure SQL Database.

**Describe Azure Synapse Analytics dedicated SQL pools.** Azure Synapse Analytics dedicated SQL pools is an MPP relational database offering in Azure that is designed for large-scale data warehouses. It uses a scale-out architecture that separates the computational engine and where data is stored so that it can efficiently process big data workloads. Data warehouse practitioners can choose how their data is distributed by configuring their tables to use hash distribution or round-robin distribution or replicating to the first distribution of each compute node.

**Describe open-source options in Azure such as Azure Database for PostgreSQL, Azure Database for MariaDB, and Azure Database for MySQL.** Organizations can migrate their existing PostgreSQL, MariaDB, and MySQL footprints to Azure to offload many of the management requirements that are associated with an on-premises environment. Like Azure SQL, Azure manages common tasks such as upgrades, patches, database backups, high availability, and threat protection for these databases without requiring any user intervention. Each of these options includes three service tiers that are designed to meet different performance requirements.

**Describe automated deployment options for relational databases in Azure.** While organizations can manually deploy relational databases in Azure through the Azure Portal, it is common to use a technology such as a scripting language or an Infrastructure as Code template to automate database deployments. Automation technologies specific to Azure include Azure PowerShell, Azure CLI, and ARM templates.

**Describe database security components.** Security for relational databases hosted in Azure can be broken down into four categories: network isolation, access management, data encryption and obfuscation, and security management. It's important to understand the different methods and technologies that are available in each of these categories. Also, be able to distinguish between authentication and authorization for access management.

**Describe common relational database management tools.** There are several tools available for database administrators and developers to build and manage database solutions. The three most common tools available from Microsoft are SQL Server Management Studio (SSMS), Azure Data Studio, and sqlcmd.

**Describe DDL and DML commands.** Standard ANSI SQL commands can be broken down into two categories: Data Definition Language (DDL) and Data Manipulation Language (DML). DDL commands refer to SQL operations used to create new objects, modify existing ones, or remove objects that are no longer required. DDL commands include CREATE, ALTER, and DROP. DML commands refer to SQL operations that read and write data in a database. DML commands include SELECT, INSERT, UPDATE, and DELETE.

**Describe techniques used to query relational data in Azure SQL, Azure Database for PostgreSQL, and Azure Database for MySQL.** SQL is a highly flexible language that allows developers to perform operations that retrieve data from multiple tables in the same query, filter data, and perform computations on data. Different SQL dialects, such as T-SQL, have several built-in functions that can be used to perform calculations and manipulate data retrieved by a query. While most SQL dialects use the same core functionality, there are some subtle differences, such as the way T-SQL and other SQL dialects limit the number of rows returned in a query.

## Review Questions

1. Which of the following isolation levels may result in queries running into the nonrepeatable read issue?
   - **A.** Snapshot
   - **B.** Read Committed
   - **C.** Repeatable Read
   - **D.** Serializable
2. Is the italicized portion of the following statement true, or does it need to be replaced with one of the other fragments that appear below? "The best practice for storing data in analytical systems such as data warehouses and OLAP models is to *normalize data using 2NF*."
   - **A.** Normalize data using 1NF
   - **B.** Normalize data using 3NF
   - **C.** De-normalize the data and use a star schema
   - **D.** No change necessary
3. You are the database administrator for a company that sells bicycles. One of the developers at the company has expressed a concern with the performance of queries that perform specific filters on columns that are not the primary key. Which of the following types of indexes should you use to increase the performance of these queries?
   - **A.** Nonclustered index
   - **B.** Clustered index
   - **C.** Clustered columnstore index
   - **D.** Filtered index
4. Is the italicized portion of the following statement true, or does it need to be replaced with one of the other fragments that appear below? "Azure SQL Managed Instance is an example of a *PaaS* solution."
   - **A.** IaaS
   - **B.** SaaS
   - **C.** DbaaS
   - **D.** No change necessary
5. You are a consultant for a company that is moving many of its applications from on-premises infrastructure to Azure. Each database must be hosted on a database platform that requires the least amount of administrative effort. One of the applications has a relatively short migration timeline, greatly reducing the amount of time to update deprecated features or fix

any compatibility issues. The databases serving this application run on SQL Server 2019 and have a few SQL Server Agent jobs. Which of the following database offerings provides the fastest time to Azure while maintaining the administrative requirement?

- **A.** Azure SQL MI
- **B.** SQL Server on Azure VM
- **C.** Azure SQL Database Single Database
- **D.** Azure SQL Database Elastic Pool

6. Which Azure virtual machine category is the recommended choice for most SQL Server workloads?
   - **A.** Compute optimized
   - **B.** Memory optimized
   - **C.** Storage optimized
   - **D.** General purpose

7. As the lead database administrator for a large retail company, you have been tasked with designing the HADR strategy for your SQL Server on Azure VM footprint. While every database needs to be replicated to a separate server in the same region, only some databases need to be replicated to another server in a different region. During a planned outage, the databases should immediately failover to the server in the same region. If there is a datacenter-wide outage, then the database will need to immediately failover to the server in the other region. Which of the following HADR options should you recommend for this approach?
   - **A.** SQL Server Database Mirroring
   - **B.** SQL Server Always On availability groups
   - **C.** Azure Site Recovery
   - **D.** SQL Server Failover Cluster Instances

8. Is the italicized portion of the following statement true, or does it need to be replaced with one of the other fragments that appear below? "A *virtual network* is a networking service in Azure that provides network isolation for relational database services such as SQL Server on Azure VM and Azure SQL MI."
   - **A.** Route table
   - **B.** Azure ExpressRoute
   - **C.** Network security group
   - **D.** No change necessary

9. Which of the following components are deployed with Azure SQL MI and exposed to the end user?
   - **A.** Network security group
   - **B.** SQL database
   - **C.** VNet
   - **D.** Virtual machine

**10.** You are designing a database solution in Azure that will serve as the storage engine for an OLTP workload. The chosen database option will need to minimize the amount of administrative effort. Along with supporting several write operations, the solution will also be used by many analysts who will be reading data to build daily summaries of the data. The solution will need to route these read-only workloads so that they do not affect the performance of the write operations. Which of the following is the best option for this scenario?

- **A.** Design the solution to use a SQL Server on Azure VM as the database solution. Create multiple virtual machines and cluster them so that they can participate in an availability group for HADR. Enable one of the instances to serve read-only workloads and set the application the analysts are using to use a read-only application intent. This will route their queries to the read-only secondary.
- **B.** Design the solution to use Azure SQL MI as the database solution. Choose the Business Critical tier and set the application the analysts are using to use a read-only application intent. This will route their queries to the read-only secondary that is deployed with the Business Critical tier of Azure SQL MI.
- **C.** Design the solution to use Azure SQL MI as the database solution. Choose the General Purpose tier and set the application the analysts are using to use a read-only application intent. This will route their queries to the read-only secondary that is deployed with the General Purpose tier of Azure SQL MI.
- **D.** Design the solution to use Azure SQL Database as the database solution. Choose the Standard tier and set the application the analysts are using to use a read-only application intent. This will route their queries to the read-only secondary that is deployed with the Standard tier of Azure SQL Database.

**11.** What is the maximum number of user databases that can be deployed to a single Azure SQL MI?

- **A.** 1,000
- **B.** 50
- **C.** 100
- **D.** 500

**12.** Which of the following services allow users to attach a private IP address to an Azure SQL Database logical server?

- **A.** VNet
- **B.** Network Security Group
- **C.** Private Link
- **D.** Azure Firewall

13. You are designing a data warehouse with an Azure Synapse Analytics dedicated SQL pool that analysts will use to power their reporting dashboards. One of the requirements is that users must be able to query data without needing to write complex T-SQL code to retrieve data. These result sets will perform aggregations over a common set of tables. Which object native to dedicated SQL pools is the most performant option for this use case?

    **A.** View

    **B.** Materialized view

    **C.** Temporary tables

    **D.** Stored procedure

14. Azure Synapse Analytics dedicated SQL pool shards data into how many distributions when performing computations?

    **A.** 30

    **B.** 100

    **C.** 45

    **D.** 60

15. You are the lead DBA of an organization that hosts its mission-critical OLTP databases on Azure SQL MI. The development team for an application using one of these databases has requested an older copy of the database be placed on a separate Azure SQL MI to perform tests on data that has since been deleted. They are asking for a version that is two days older than the current date. Which of the following options should you use to copy the database to the other Azure SQL MI?

    **A.** Azure Blob storage automatically stores Azure SQL MI database backups. Use a RESTORE FROM URL command to restore a backup of the database from two days ago on the second Azure SQL MI.

    **B.** Azure manages backups for Azure SQL MI with automated backups. Perform a point-in-time restore to restore a backup of the database from two days ago on the second Azure SQL MI.

    **C.** Azure manages backups for Azure SQL MI with automated backups. Perform a point-in-time restore to restore a backup of the database from two days ago on the first Azure SQL MI using a different name. Then, configure transactional replication to replicate the restored database from the first Azure SQL MI to the second.

    **D.** Azure manages backups for Azure SQL MI with automated backups. Perform a point-in-time restore to restore a backup of the database from two days ago on the first Azure SQL MI using a different name. Then, use Azure Data Synch to replicate the restored database from the first Azure SQL MI to the second.

16. The General Purpose service tier that is available for Azure SQL Database uses which purchasing model?

    **A.** vCore-based

    **B.** DTU-based

    **C.** DWU-based

    **D.** DBU-based

17. Is the italicized portion of the following statement true, or does it need to be replaced with one of the other fragments that appear below? "*Azure Database for MySQL* offers three deployment models, including a Hyperscale deployment model that uses a scale-out architecture to support large OLTP workloads."
    - **A.** Azure Database for MariaDB
    - **B.** Azure Database for PostgreSQL
    - **C.** Azure Database for HBase
    - **D.** No change necessary

18. When running an Azure PowerShell script from a local machine or VM, which of the following Azure PowerShell commands must be run at the beginning of a script to establish a connection with an Azure environment?
    - **A.** Connect-AzSession
    - **B.** Connect-AzAccount
    - **C.** Connect-AzureRmAccount
    - **D.** Connect-AzureRmSession

19. You are designing an Azure Synapse Analytics dedicated SQL pool data warehouse that will be used to serve as the single source of truth for an e-commerce company. There are several large fact tables, each of which includes columns that are used to identify each row. These columns have many distinct values. Which of the following table distribution designs is most appropriate for these fact tables?
    - **A.** Round-robin
    - **B.** Replicated table
    - **C.** Hash
    - **D.** Broadcast

20. Which of the following resource provider and resource type combinations is used by Azure to manage the deployment of an Azure SQL Database?
    - **A.** Microsoft.Sql/managedInstances
    - **B.** Microsoft.Sql/databases
    - **C.** Microsoft.Sql/servers
    - **D.** Microsoft.Sql/servers/databases

21. What Azure PowerShell command can be used to create a new Azure SQL Database?
    - **A.** New-AzSqlDatabase
    - **B.** Create-AzSqlDatabase
    - **C.** New-AzRmSqlDatabase
    - **D.** Create-AzRmSqlDatabase

**22.** Which of the following RBAC roles lets a user create and manage databases without giving them access? Choose the option that gives the user the least number of privileges.

- **A.** Azure Active Directory Administrator
- **B.** SQL DB Contributor
- **C.** SQL Security Manager
- **D.** Contributor

**23.** You are designing an application that will be serving highly sensitive information to a web application. The data must be encrypted so that the only the application can decrypt the data, preventing database administrators from being able to view the raw values. Which of the following options is the best choice for encrypting the columns storing this sensitive information?

- **A.** Dynamic Data Masking
- **B.** Denying access to the columns using a `DENY` statement
- **C.** Always Encrypted
- **D.** Transparent Data Encryption

**24.** Which database management tool is most suited for performing administrative tasks such as managing user permissions, optimizing queries, and building HADR solutions?

- **A.** Azure Data Studio
- **B.** Azure PowerShell
- **C.** Azure Portal
- **D.** SQL Server Management Studio

**25.** You are the database administrator for a large e-commerce company. One of the developers is having issues connecting to one of the Azure SQL Databases you manage and has come to you for help. The error message in SSMS indicates a login failure and states that the user cannot open the "master" database. You come to learn that the developer is logging into the database using a database contained user. What step should you tell the developer to take to remediate the issue?

- **A.** Change the database context on the login screen in SSMS from *default* to the database the user has access to.
- **B.** Use the SQL administrator credentials to log into the logical server to access the database.
- **C.** Use an AAD account to log into the database.
- **D.** Enable TCP/IP for the SQL instance.

**26.** Which category of SQL statements does a `CREATE TABLE` statement fall under?

- **A.** DML
- **B.** DDL
- **C.** DCL
- **D.** TCL

**27.** What component of Azure Defender for SQL will alert users to malicious activities such as SQL injection and data exfiltration?

**A.** SQL auditing

**B.** Vulnerability assessment

**C.** SQL Server extended events

**D.** Advanced Threat Protection

**28.** What T-SQL statement can be used to add an Azure Active Directory user or group as a user for an Azure SQL Database?

**A.** CREATE EXTERNAL PROVIDER AAD; CREATE USER [<AAD_User>] FROM AAD;

**B.** CREATE USER [<AAD_User>] FROM EXTERNAL PROVIDER;

**C.** CREATE USER [<AAD_User>] FROM EXTERNAL SERVICE;

**D.** CREATE USER [<AAD_User>]

**29.** Which of the following steps comes first in the logical processing order of a T-SQL SELECT statement?

**A.** FROM

**B.** GROUP BY

**C.** ORDER BY

**D.** SELECT

**30.** Which of the following is a Unicode data type that is used to define string data that has variable size?

**A.** VARCHAR()

**B.** NVARCHAR()

**C.** CHAR()

**D.** NCHAR()

**31.** Is the italicized portion of the following statement true, or does it need to be replaced with one of the other fragments that appear below? "A(n) *Full Join* is a join type that is used to retrieve data from both tables that meets the join condition."

**A.** Full inner join

**B.** Inner join

**C.** Outer join

**D.** No change necessary